图灵原创

资深Android开发者，CSDN认证专家、
超人气博主郭霖最新力作!



第一行代码 Android

第2版

郭 霖◎著

基于Android 7.0 / 难点、疑点透彻讲解 / 通俗易懂无废话 / 入门到上架APP一本通

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

数字版权声明

作者简介

郭霖

Android软件开发工程师。从事Android开发工作6年，有着丰富的项目实战经验，负责及参与开发过多款移动应用与游戏，对Android系统架构及应用层开发有着深入的理解。

2013年3月开始，在CSDN上发表Android技术相关博文，很快获得了大量网友的好评。目前博客访问量已超过500万次，评论超过10000条。荣获CSDN认证专家，并被连续评选为CSDN 2013、2014、2015年度博客之星。

博客：http://guolin.tech

个人微信公众号

每天推送技术文章





第一行代码

Android

第 2 版

郭 霖◎著

图书在版编目（CIP）数据

第一行代码——Android / 郭霖著. -- 2版. -- 北京 : 人民邮电出版社, 2016.12（2017.5重印）
（图灵原创）
ISBN 978-7-115-43978-9

Ⅰ. ①第… Ⅱ. ①郭… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2016)第267736号

内 容 提 要

本书被广大 Android 开发者誉为“Android 学习第一书”。全书系统全面、循序渐进地介绍了 Android 软件开发的必备知识、经验和技巧。

第 2 版基于 Android 7.0 对第 1 版进行了全面更新，将所有知识点都在最新的 Android 系统上进行重新适配，使用全新的 Android Studio 开发工具代替之前的 Eclipse，并添加了对 Material Design、运行时权限、多窗口模式、Gradle、RecyclerView、百分比布局、OkHttp、Lambda 表达式等全新知识点的详细讲解。

本书内容通俗易懂，由浅入深，既是 Android 初学者的入门必备，也是 Android 开发者的进阶首选。

◆ 著　　　　郭　霖
责任编辑　王军花
执行编辑　张　霞
责任印制　彭志环
封面插画　巫俊武
封面设计　潘建永　陈　冰

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　印刷

◆ 开本：800×1000　1/16
印张：36.25
字数：856千字　　　　2016年 12 月第 2 版
印数：144 001 – 149 000册　　2017年 5 月北京第 6 次印刷

定价：79.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广字第 8052 号

前　　言

虽然我从事Android开发工作已经很多年了，但是之前从来没有想过自己要去写一本Android技术相关的书。在我看来，写一本书可以算是一个很庞大的工程，写一本好书的难度并不亚于开发一款好的应用程序。

由于我长期坚持在CSDN上发表技术博文，因而得到了大量网友的认可，也积累了一定的名气。很荣幸的是，人民邮电出版社图灵公司的前副总编辑陈冰老师联系上了我，希望我可以写一本关于Android开发技术的书，这着实让我受宠若惊。

在写本书第1版的时候，我可以说是费了相当大的心思。写书和写博客最大的区别在于，书的内容不能像博客那样散乱，不能想到哪里写到哪里，而是一定要系统化，要循序渐进，基本上在写第1章的时候就应该把全书的内容都确定下来了。

令我非常欣慰的是，本书的第1版在推出之后获得了广大读者的强烈认可，在短短两年时间内，已经成为了国内最畅销的Android技术书。各大书店、图书馆都能看到《第一行代码》的身影，许多学校和培训机构也纷纷将《第一行代码》选为Android课程的教材。

不过，在科技高速发展的今天，各种技术的发展都是日新月异的。在两年的时间里，Android操作系统经历了5.0、6.0、7.0的飞速升级。不可否认的是，本书第1版中的不少知识点都已经过时，而且这两年间出现的很多新知识，第1版中也没有涵盖。因此，这让我坚定了写作本书第2版的想法。

刚开始写的时候，我以为只是小修小补，但事实上并没有我想象得那么轻松。除了介绍新知识点以外，书中之前的所有项目都需要重新编写和测试，以保证代码在新老系统上的兼容性。另外，由于Android从5.0系统开始，UI风格变化很大，因此第1版中所有的截图都需要更新。毫不夸张地说，我几乎重写了整本书。

而现在，你手中捧着的正是全新版的《第一行代码》，同时这也是国内第一本基于Android 7.0系统写作的技术书。我真诚地希望你可以用心去阅读这本书，因为每多掌握一份知识，你就会多一份喜悦。Enjoy it!

第 2 版的变化

由于第2版修改内容繁多，因此这里我只列举出最主要的变化。首先是开发工具上的改变，本书第1版使用的开发工具是Eclipse，而第2版使用了目前最新的Android Studio 2.2版本。另外，

本书第1版是基于Android 4.x系统的，而第2版是基于Android 7.0系统的，其中囊括了新系统中的诸多知识点，包括Android 5.0系统中引入的Material Design、Android 6.0系统中引入的运行时权限和Doze模式、Android 7.0系统中引入的多窗口模式等。

除此之外，第2版还加入了Gradle、RecyclerView、百分比布局、OkHttp、Lambda表达式等全新知识点的讲解，内容将前所未有地充实。

读者对象

本书内容通俗易懂，由浅入深，既适合初学者阅读，也同样适合专业人员。学习本书内容之前，你并不需要有任何的Android基础，但是你需要有一定的Java基础，因为Android开发都是使用Java语言的，而本书并不会去专门介绍Java方面的知识。

阅读本书时，你可以根据自身的情况来决定如何阅读。如果你是初学者的话，建议你从第1章开始循序渐进地阅读，这样理解起来就不会感到吃力。而如果你已经有了一定的Android基础，那么就可以选择某些你感兴趣的章节进行跳跃式的阅读。但请记住，很多章最后的最佳实践部分一定是你不想错过的。

本书内容

正如前面所说，本书的内容是非常系统化的，不仅全面介绍了那些你必须掌握的知识，而且保证了各章的难度都是梯度式上升的。全书一共分为15章，涵盖了四大组件、UI、碎片、数据存储、多媒体、网络、定位服务等方方面面的知识。为了让你在学完所有内容之后还可以有综合运用的能力，本书的尾声部分还会带你一起开发一个天气预报程序，并教会你如何将程序发布到应用商店，以及如何在程序中嵌入广告盈利。

除此之外，本书的第5章、第7章、第11章、第14章中都穿插有对Git的讲解，如果想要掌握它的用法，这几章的内容是绝对不能错过的。

本书中各个章节的内容都相对比较独立，因此除了可以循序渐进地学习之外，你还可以把它当成一本参考手册，随时查阅。

源码下载

首先，我建议你在学习本书的时候将所有项目的源码都亲手敲上一遍，因为只有这样才能加深你对代码的理解。不过为了方便于你的学习，我还是提供了书中所有项目的源码，请仅在需要的时候再去参考（如下载项目中的图片资源）。切勿直接将源码复制粘贴就当成自己的东西了，只有亲手敲过的代码才真正是你自己的。

源码下载地址：https://github.com/guolindev/booksource。

致　　谢

在这近一年的时间里，我又完成了一项浩大的工程。和写作本书第1版时的感觉类似，当全书完稿之后，回顾整本书，我仍然不敢相信这所有的内容竟然是我一字字地敲出来的。

如今这已经是我写的第二本书了，和写第一本书时的情况不同，现在我有了更广的人脉和资源，有了更多的人愿意帮助和支持我来完成一本更好的技术书。因此，我要在这里对很多人表示感谢。

首先我要感谢我的父母，感谢你们将我抚养长大，感谢你们的付出，让我从小不用为生计、上学而发愁，可以一直做我自己想做的事情，也感谢你们指引我走上了技术这条路。

其次我要感谢我的妻子，感谢你每天为我准备好一日三餐，感谢你对我永远的包容，不管是平日的加班还是没日没夜的写书，你都一直默默地理解和支持我。

我还非常感谢本书第1版的编辑陈冰老师，如果没有你当初在CSDN上找到我，并邀请我写书，就不会有现在的《第一行代码》。另外，你也是当时唯一一个坚信这本书一定会大卖的人，甚至连我自己当时都没有如此的眼光。

我也非常感谢本书第2版的编辑张霞，你全程负责了第2版的出版工作，并且完成得非常出色。你对文字的把控能力让我敬佩，感谢你对书中每一章节的尽心审阅，才能让这本书更趋近于完美。

另外我还要特别感谢一部分人，你们在对本书的内容建议、勘误检查、代码纠错，甚至是对我个人的支持等方面都作出了卓越的贡献。有了你们的帮助，才会有这样一本更加出色的书呈现在所有人面前，这本书上也理应有你们的名字（按姓氏拼音排序，排名不分先后）：

陈建林　陈俊杰　陈　雷　陈　龙　陈　琪　陈秀相　陈逸鸣　代云蛟　董霖轩　段郭森

高鹤泉　高太稳　关爱民　何以诚　胡恩泽　黄　楠　赖　帆　李济洲　李建友　李沛明

李　潭　李永鹏　李志云　林火荣　刘　萌　刘明渊　刘治国　陆德俊　罗亚超　吕国鑫

马文杰　覃文斌　孙建飞　王柏强　王光东　王　杰　王　龙　王路路　王　鹏　王荣宗

王善昌　韦振南　吴　波　吴宏权　吴绍志　徐　阳　轩仲宽　杨　辉　易静杰　查　童

张鸿洋　张英祥　赵翠龙　赵庆元　赵迎超　郑传书　郑敏馨　庄育锋　周　苏　朱海丰

目　　录

第 1 章

开始启程——你的第一行 Android 代码

欢迎你来到 Android 世界！Android 系统是目前世界上市场占有率最高的移动操作系统，不管你在哪里，都可以看到 Android 手机几乎无处不在。今天的 Android 世界可谓欣欣向荣，可是你知道它的过去是什么样的吗？我们一起来看一看它的发展史吧。

2003 年 10 月，Andy Rubin 等人一起创办了 Android 公司。2005 年 8 月谷歌收购了这家仅仅成立了 22 个月的公司，并让 Andy Rubin 继续负责 Android 项目。在经过了数年的研发之后，谷歌终于在 2008 年推出了 Android 系统的第一个版本。但自那之后，Android 的发展就一直受到重重阻挠。乔布斯自始至终认为 Android 是一个抄袭 iPhone 的产品，里面剽窃了诸多 iPhone 的创意，并声称一定要毁掉 Android。而本身就是基于 Linux 开发的 Android 操作系统，在 2010 年被 Linux 团队从 Linux 内核主线中除名。又由于 Android 中的应用程序都是使用 Java 开发的，甲骨文则针对 Android 侵犯 Java 知识产权一事对谷歌提起了诉讼……

可是，似乎再多的困难也阻挡不了 Android 快速前进的步伐。由于谷歌的开放政策，任何手机厂商和个人都能免费获取到 Android 操作系统的源码，并且可以自由地使用和定制。三星、HTC、摩托罗拉、索爱等公司都推出了各自系列的 Android 手机，Android 市场上百花齐放。仅仅推出两年后，Android 就超过了已经霸占市场逾十年的诺基亚 Symbian，成为了全球第一大智能手机操作系统，并且每天都还会有数百万台新的 Android 设备被激活。而近几年，国内的手机厂商也是大放异彩，小米、华为、魅族等新兴品牌都推出了相当不错的 Android 手机，并且也获得了市场的广泛认可，目前 Android 已经占据了全球智能手机操作系统 70%以上的份额。

说了这些，想必你已经体会到 Android 系统炙手可热的程度，并且迫不及待地想要加入到 Android 开发者的行列当中了吧。试想一下，十个人中有七个人的手机都可以运行你编写的应用程序，还有什么能比这个更诱人的呢？那么从今天起，我就带你踏上学习 Android 的旅途，一步步地引导你成为一名出色的 Android 开发者。

好了，现在我们就来一起初窥一下 Android 世界吧。

1.1　了解全貌——Android 王国简介

Android 从面世以来到现在已经发布了二十几个版本了。在这几年的发展过程中，谷歌为 Android 王国建立了一个完整的生态系统。手机厂商、开发者、用户之间相互依存，共同推进着 Android 的蓬勃发展。开发者在其中扮演着不可或缺的角色，因为如果没有开发者来制作丰富的应用程序，那么不管多么优秀的操作系统，也是难以得到大众用户喜爱的，相信没有多少人能够忍受没有 QQ、微信的手机吧。而且，谷歌推出的 Google Play 更是给开发者带来了大量的机遇，只要你能制作出优秀的产品，在 Google Play 上获得了用户的认可，你就完全可以得到不错的经济回报，从而成为一名独立开发者，甚至是成功创业！

那我们现在就以一个开发者的角度，去了解一下这个操作系统吧。纯理论型的东西也比较无聊，怕你看睡着了，因此我只挑重点介绍，这些东西跟你以后的开发工作都是息息相关的。

1.1.1　Android 系统架构

为了让你能够更好地理解 Android 系统是怎么工作的，我们先来看一下它的系统架构。Android 大致可以分为四层架构：Linux 内核层、系统运行库层、应用框架层和应用层。

1. Linux 内核层

Android 系统是基于 Linux 内核的，这一层为 Android 设备的各种硬件提供了底层的驱动，如显示驱动、音频驱动、照相机驱动、蓝牙驱动、Wi-Fi 驱动、电源管理等。

2. 系统运行库层

这一层通过一些 C/C++库来为 Android 系统提供了主要的特性支持。如 SQLite 库提供了数据库的支持，OpenGL|ES 库提供了 3D 绘图的支持，Webkit 库提供了浏览器内核的支持等。

同样在这一层还有 Android 运行时库，它主要提供了一些核心库，能够允许开发者使用 Java 语言来编写 Android 应用。另外，Android 运行时库中还包含了 Dalvik 虚拟机（5.0 系统之后改为 ART 运行环境），它使得每一个 Android 应用都能运行在独立的进程当中，并且拥有一个自己的 Dalvik 虚拟机实例。相较于 Java 虚拟机，Dalvik 是专门为移动设备定制的，它针对手机内存、CPU 性能有限等情况做了优化处理。

3. 应用框架层

这一层主要提供了构建应用程序时可能用到的各种 API，Android 自带的一些核心应用就是使用这些 API 完成的，开发者也可以通过使用这些 API 来构建自己的应用程序。

4. 应用层

所有安装在手机上的应用程序都是属于这一层的，比如系统自带的联系人、短信等程序，或者是你从 Google Play 上下载的小游戏，当然还包括你自己开发的程序。

结合图 1.1 你将会理解得更加深刻，图片源自维基百科。

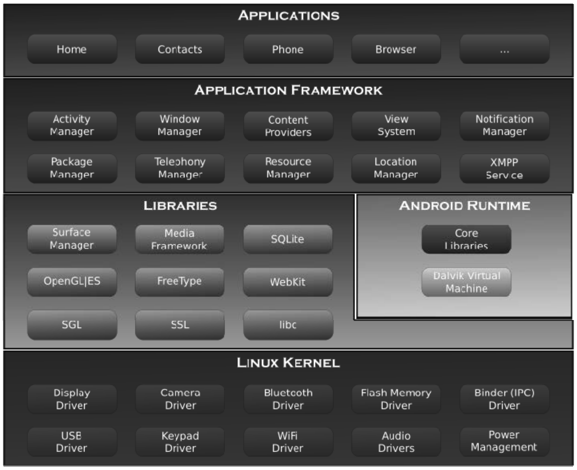


图 1.1　Android 系统架构

1.1.2　Android 已发布的版本

2008 年 9 月，谷歌正式发布了 Android 1.0 系统，这也是 Android 系统最早的版本。随后的几年，谷歌以惊人的速度不断地更新 Android 系统，2.1、2.2、2.3 系统的推出使 Android 占据了大量的市场。2011 年 2 月，谷歌发布了 Android 3.0 系统，这个系统版本是专门为平板电脑设计的，但也是 Android 为数不多的比较失败的版本，推出之后一直不见什么起色，市场份额也少得可怜。不过很快，在同年的 10 月，谷歌又发布了 Android 4.0 系统，这个版本不再对手机和平板进行差异化区分，既可以应用在手机上，也可以应用在平板上。2014 年 Google I/O 大会上，谷歌推出了号称史上版本改动最大的 Android 5.0 系统，其中使用 ART 运行环境替代了 Dalvik 虚拟机，大大提升了应用的运行速度，还提出了 Material Design 的概念来优化应用的界面设计。除此之外，还推出了 Android Wear、Android Auto、Android TV 系统，从而进军可穿戴设备、汽车、电视等全新领域。之后 Android 的更新速度更加迅速，2015 年 Google I/O 大会上推出了 Android 6.0 系统，加入运行时权限功能，2016 年 Google I/O 大会上推出了 Android 7.0 系统，加入多窗口模式功能，这也是目前最新的 Android 系统版本。

下表列出了目前主要的 Android 系统版本及其详细信息。你看到这张表格时，数据可能已经发生了变化，查看最新的数据可以访问 http://developer.android.google.cn/about/dashboards/。

版　本　号	系统代号	API	市场占有率
2.2	Froyo	8	0.1%
2.3.3 – 2.3.7	Gingerbread	10	1.5%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	1.3%
4.1.x	Jelly Bean	16	5.6%
4.2.x		17	7.7%
4.3		18	2.3%

（续）

版　本　号	系统代号	API	市场占有率
4.4	KitKat	19	27.7%
5.0	Lollipop	21	13.1%
5.1		22	21.9%
6.0	Marshmallow	23	18.7%
7.0	Nougat	24	0.1%

从上表中可以看出，目前 4.0 以上的系统已经占据了超过 98%的 Android 市场份额，因此我们本书中开发的程序也只面向 4.0 以上的系统，2.x 的系统就不再去兼容了。

1.1.3　Android 应用开发特色

预告一下，你马上就要开始真正的 Android 开发旅程了。不过先别急，在开始之前我们再来一起看一看，Android 系统到底提供了哪些东西，可供我们开发出优秀的应用程序。

1. 四大组件

Android 系统四大组件分别是活动（Activity）、服务（Service）、广播接收器（Broadcast Receiver）和内容提供器（Content Provider）。其中活动是所有 Android 应用程序的门面，凡是在应用中你看得到的东西，都是放在活动中的。而服务就比较低调了，你无法看到它，但它会一直在后台默默地运行，即使用户退出了应用，服务仍然是可以继续运行的。广播接收器允许你的应用接收来自各处的广播消息，比如电话、短信等，当然你的应用同样也可以向外发出广播消息。内容提供器则为应用程序之间共享数据提供了可能，比如你想要读取系统电话簿中的联系人，就需要通过内容提供器来实现。

2. 丰富的系统控件

Android 系统为开发者提供了丰富的系统控件，使得我们可以很轻松地编写出漂亮的界面。当然如果你品位比较高，不满足于系统自带的控件效果，也完全可以定制属于自己的控件。

3. SQLite 数据库

Android 系统还自带了这种轻量级、运算速度极快的嵌入式关系型数据库。它不仅支持标准的 SQL 语法，还可以通过 Android 封装好的 API 进行操作，让存储和读取数据变得非常方便。

4. 强大的多媒体

Android 系统还提供了丰富的多媒体服务，如音乐、视频、录音、拍照、闹铃，等等，这一切你都可以在程序中通过代码进行控制，让你的应用变得更加丰富多彩。

5. 地理位置定位

移动设备和 PC 相比起来，地理位置定位功能应该可以算是很大的一个亮点。现在的 Android 手机都内置有 GPS，走到哪儿都可以定位到自己的位置，发挥你的想象就可以做出创意十足的应

用，如果再结合功能强大的地图功能，LBS 这一领域潜力无限。

既然有 Android 这样出色的系统给我们提供了这么丰富的工具，你还用担心做不出优秀的应用吗？好了，纯理论的东西就介绍到这里，我知道你已经迫不及待想要开始真正的开发之旅了，那我们就开始启程吧！

1.2　手把手带你搭建开发环境

俗话说得好，“工欲善其事，必先利其器”，开着记事本就想去开发 Android 程序显然不是明智之举，选择一个好的 IDE 可以极大幅度地提高你的开发效率，因此本节我就将手把手带着你把开发环境搭建起来。

1.2.1　准备所需要的工具

我现在对你了解还并不多，但我希望你已经是一个颇有经验的 Java 程序员，这样你理解本书的内容时将会轻而易举，因为 Android 程序都是使用 Java 语言编写的。如果你对 Java 只是略有了解，那阅读本书应该会有一点困难，不过一边阅读一边补充 Java 知识也是可以的。但如果你对 Java 完全没有了解，那么我建议你可以暂时将本书放下，先买本介绍 Java 基础知识的书学上两个星期，把 Java 的基本语法和特性都学会了，再来继续阅读这本书。

好了，既然你已经阅读到这里，说明你已经掌握 Java 的基本用法了，下面我们就来看一看开发 Android 程序需要准备哪些工具。

- JDK。JDK 是 Java 语言的软件开发工具包，它包含了 Java 的运行环境、工具集合、基础类库等内容。需要注意的是，本书中的 Android 程序必须要使用 JDK 8 或以上版本才能进行开发。
- Android SDK。Android SDK 是谷歌提供的 Android 开发工具包，在开发 Android 程序时，我们需要通过引入该工具包，来使用 Android 相关的 API。
- Android Studio。在很早之前，Android 项目都是用 Eclipse 来开发的，相信所有 Java 开发者都一定会对这个工具非常熟悉，它是 Java 开发神器，安装 ADT 插件后就可以用来开发 Android 程序了。而在 2013 年的时候，谷歌推出了一款官方的 IDE 工具 Android Studio，由于不再是以插件的形式存在，Android Studio 在开发 Android 程序方面要远比 Eclipse 强大和方便得多。不过由于 Android Studio 早期的测试版本并不是非常稳定，所以本书的第 1 版仍然选用 Eclipse 来作为开发工具。而如今，Android Studio 已经推出了 2.2 版本，稳定性完全不再是问题，普及程度方面也远超 Eclipse，没有比现在更适合的时机来换用 Android Studio 了，因此本书中所有的代码都将在 Android Studio 上进行开发。

1.2.2　搭建开发环境

当然，上述软件并不需要你去一个个地下载，因为谷歌为了简化搭建开发环境的过程，将所

有需要用到的工具都帮我们集成好了，到 Android 官网就可以下载最新的开发工具，下载地址是：https://developer.android.google.cn/studio/index.html。不过，Android 官网通常都需要科学上网才能访问，如果你无法访问的话，也可以直接到我的百度网盘去下载，下载地址是：https://pan.baidu.com/s/1nuABMDb。（注意网址中是阿拉伯数字 1，而不是英文字母 l。）

你下载下来的将是一个安装包，安装的过程也很简单，一直点击 Next 就可以了。其中选择安装组件时建议全部勾上，如图 1.2 所示。



图 1.2　选择安装组件

接下来还会让你选择 Android Studio 的安装地址以及 Android SDK 的安装地址，这些根据你自己电脑的实际情况选择就行了，不想改动的话就保持默认，如图 1.3 所示。

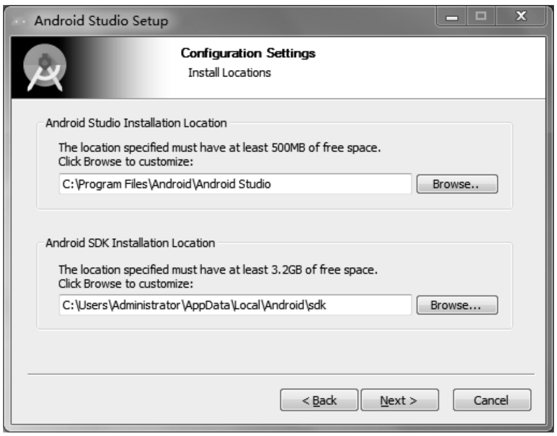


图 1.3　选择安装地址

后面就没什么需要注意的了，全部保持默认项，一直点击 Next 即可完成安装，如图 1.4 所示。



图 1.4　安装完成

现在点击 Finish 按钮来启动 Android Studio，一开始会让你选择是否导入之前 Android Studio 版本的配置，由于这是我们首次安装，这里选择不导入就可以了，如图 1.5 所示。

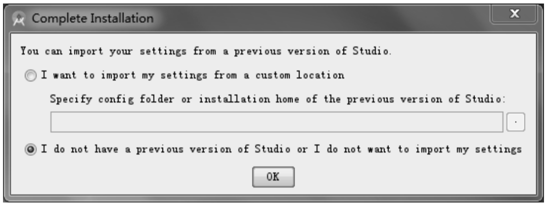


图 1.5　选择不导入配置

点击 OK 按钮会进入到 Android Studio 的配置界面，如图 1.6 所示。



图 1.6　Android Studio 的配置界面

然后点击 Next 开始进行具体的配置，如图 1.7 所示。

这里我们可以选择 Android Studio 的安装类型，有 Standard 和 Custom 两种。Standard 表示一切都使用默认的配置，比较方便；Custom 则可以根据用户的特殊需求进行自定义。简单起见，这里我们就选择 Standard 类型了，继续点击 Next 完成配置工作，如图 1.8 所示。

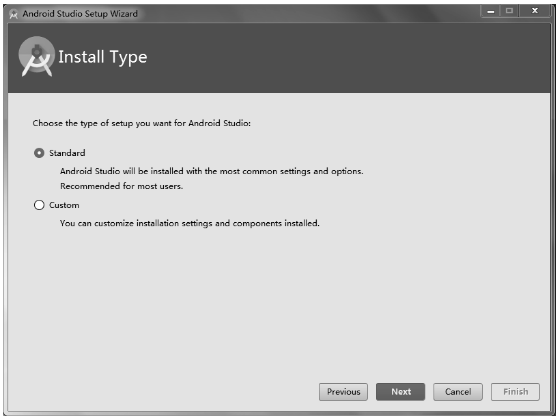


图 1.7　选择安装类型

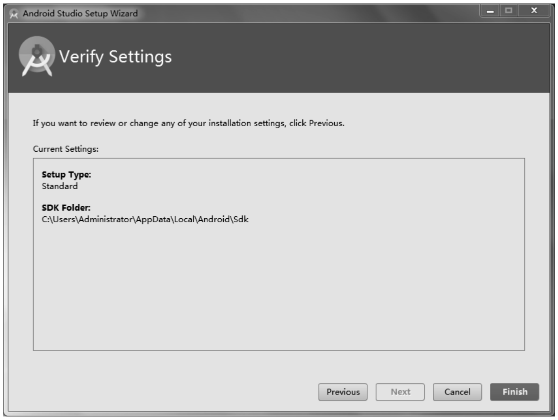


图 1.8　完成 Android Studio 配置

现在点击 Finish 按钮，配置工作就全部完成了。然后 Android Studio 会尝试联网下载一些更新，等待更新完成后再点击 Finish 按钮就会进入 Android Studio 的欢迎界面，如图 1.9 所示。



图 1.9　Android Studio 的欢迎界面

目前为止，Android 开发环境就已经全部搭建完成了。那现在应该做什么？当然是写下你的第一行 Android 代码了，让我们快点开始吧。

1.3　创建你的第一个 Android 项目

任何一个编程语言写出的第一个程序毫无疑问都会是 Hello World，这已经是自 20 世纪 70 年代一直流传下来的传统，在编程界已成为永恒的经典，那我们当然也不会搞例外了。

1.3.1　创建 HelloWorld 项目

在 Android Studio 的欢迎界面点击 Start a new Android Studio project，会打开一个创建新项目的界面，如图 1.10 所示。

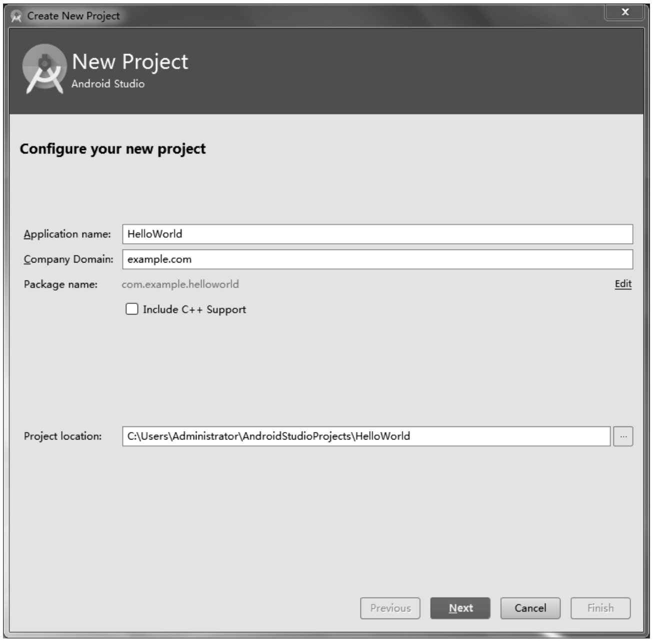


图 1.10　创建新项目

其中 Application name 表示应用名称，此应用安装到手机之后会在手机上显示该名称，这里我们填入 HelloWorld。Company Domain 表示公司域名，如果是个人开发者，没有公司域名的话，那么就像我一样填 example.com 就可以了。Package name 表示项目的包名，Android 系统就是通过包名来区分不同应用程序的，因此包名一定要具有唯一性。Android Studio 会根据应用名称和公司域名来自动帮我们生成合适的包名，如果你不想使用默认生成的包名，也可以点击右侧的 Edit 按钮自行修改。最后，Project location 表示项目代码存放的位置，如果没有特殊要求的话，这里也保持默认就可以了。

接下来点击 Next 可以对项目的最低兼容版本进行设置，如图 1.11 所示。



图 1.11　设置项目的最低兼容版本

前面已经说过，Android 4.0 以上的系统已经占据了超过 98%的 Android 市场份额，因此这里我们将 Minimum SDK 指定成 API 15 就可以了。另外，Wear、TV 和 Android Auto 这几个选项分别是用于开发可穿戴设备、电视和汽车程序的，目前这几个领域在国内还没有普及，我们暂时就先忽略吧。接着点击 Next 会跳转到创建活动界面，这里我们可以选择一种模板，如图 1.12 所示。



图 1.12　选择模板

可以看到，Android Studio 提供了很多种内置模板，不过由于我们才刚刚开始学习，用不着这么多复杂的模板，这里直接选择 Empty Activity 来创建一个空的活动就可以了。

继续点击 Next，可以给创建的活动和布局命名，如图 1.13 所示。

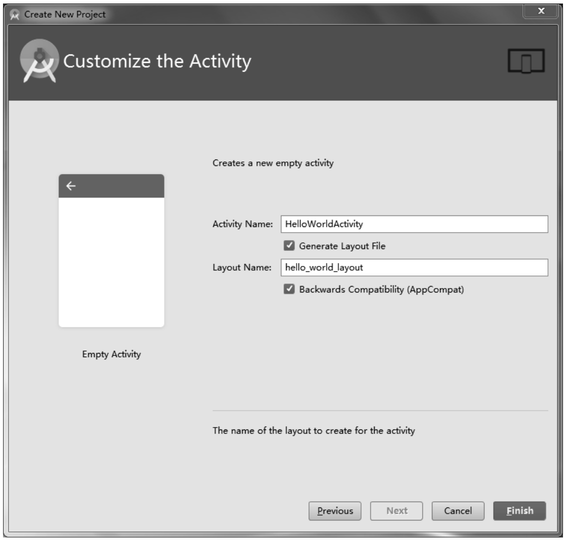


图 1.13 给活动和布局命名

其中，Activity Name 表示活动的名字，这里填入 HelloWorldActivity，Layout Name 表示布局的命名，这里填入 hello_world_layout。然后点击 Finish 按钮，并耐心等待一会儿，项目就会创建成功了，如图 1.14 所示。

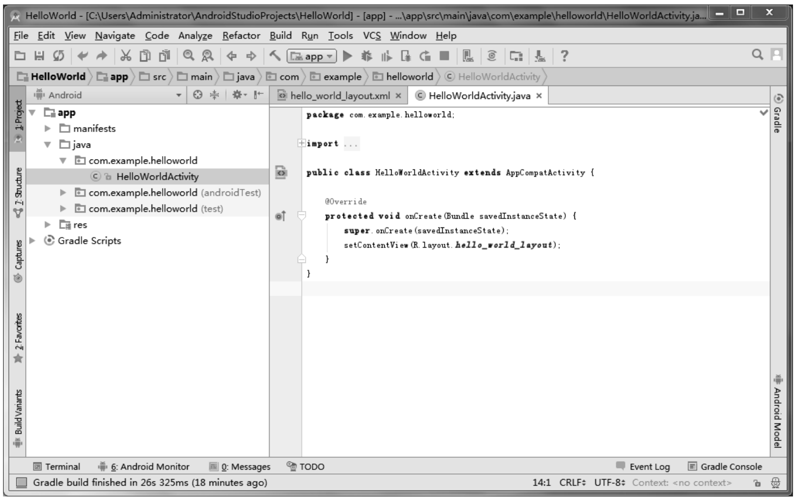


图 1.14 项目创建成功

1.3.2　启动模拟器

由于 Android Studio 自动为我们生成了很多东西，你现在不需要编写任何代码，HelloWorld 项目就已经可以运行了。但是在此之前还必须要有一个运行的载体，可以是一部 Android 手机，也可以是 Android 模拟器。这里我们暂时先使用模拟器来运行程序，如果你想立刻就将程序运行到手机上的话，可以参考 8.1 节的内容。

那么我们现在就来创建一个 Android 模拟器，观察 Android Studio 顶部工具栏中的图标，如图 1.15 所示。



图 1.15　顶部工具栏中的图标

其中，最左边的按钮就是用于创建和启动模拟器的，点击该按钮，会弹出如图 1.16 所示的窗口。



图 1.16　创建模拟器

可以看到，目前我们的模拟器列表中还是空的，点击 Create Virtual Device 按钮就可以立刻开始创建了，如图 1.17 所示。

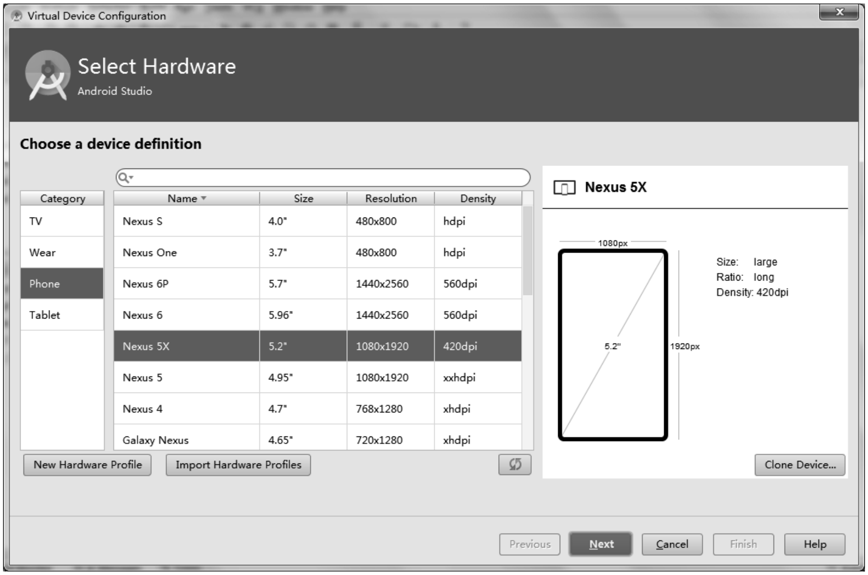


图 1.17　选择要创建的模拟器设备

这里有很多种设备可供我们选择，不仅能创建手机模拟器，还可以创建平板、手表、电视等模拟器。

那么我就选择创建 Nexus 5X 这台设备的模拟器了，点击 Next，如图 1.18 所示。

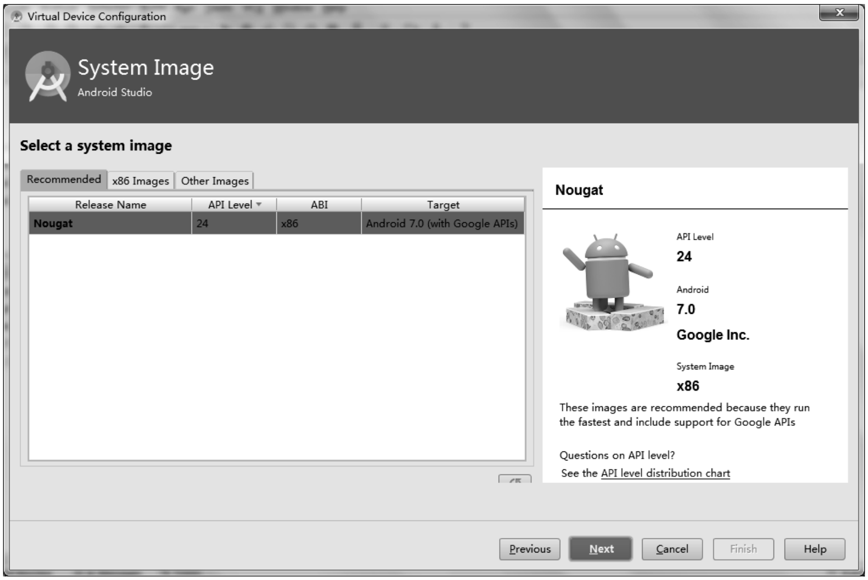


图 1.18　选择模拟器的操作系统版本

这里可以选择模拟器所使用的操作系统版本，毫无疑问，我们肯定要选择最新的 Android 7.0 系统。继续点击 Next，如图 1.19 所示。



图 1.19　确认模拟器配置

在这里我们可以对模拟器的一些配置进行确认，比如说指定模拟器的名字、分辨率、横竖屏等信息，如果没有特殊需求的话，全部保持默认就可以了。点击 Finish 完成模拟器的创建，然后会弹出如图 1.20 所示的窗口。



图 1.20　模拟器列表

可以看到，现在模拟器列表中已经存在一个创建好的模拟器设备了，点击 Actions 栏目中最左边的三角形按钮即可启动模拟器。模拟器会像手机一样，有一个开机过程，启动完成之后的界面如图 1.21 所示。



图 1.21　启动后的模拟器界面

很清新的 Android 界面出来了！看上去还挺不错吧，你几乎可以像使用手机一样使用它，Android 模拟器对手机的模仿度非常高，快去体验一下吧。

1.3.3　运行 HelloWorld

现在模拟器已经启动起来了，那么下面我们就开始将 HelloWorld 项目运行到模拟器上。观察 Android Studio 顶部工具栏中的图标，如图 1.22 所示。其中左边的锤子按钮是用来编译项目的，中间的下拉列表是用来选择运行哪一个项目的，通常 app 就是当前的主项目，右边的三角形按钮是用来运行项目的。



图 1.22　顶部工具栏中的图标

现在点击右边的运行按钮，会弹出一个选择运行设备的对话框，如图 1.23 所示。

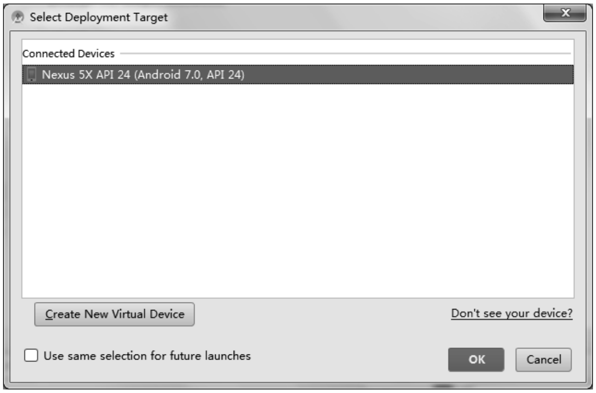


图 1.23 选择运行设备对话框

可以看到，我们刚刚创建的模拟器现在是在线的，点击 OK 按钮，稍微等待一会儿，HelloWorld 项目就会运行到模拟器上了，结果应该和图 1.24 中显示的是一样的。

HelloWorld 项目运行成功！并且你会发现，模拟器上已经安装上 HelloWorld 这个应用了。打开启动器列表，如图 1.25 所示。

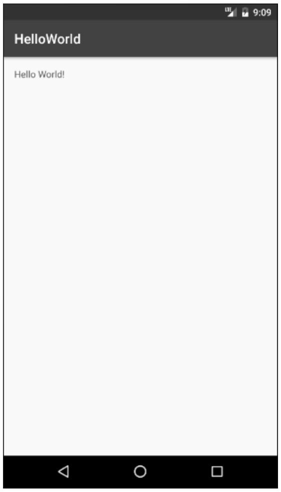


图 1.24 运行 HelloWorld 项目

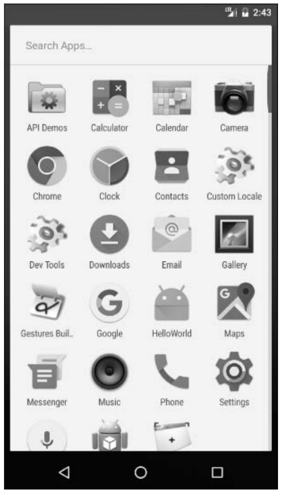


图 1.25 查看启动器列表

这个时候你可能会说我坑你了，说好的第一行代码呢？怎么一行还没写，项目就已经运行起来了？这个只能说是因为 Android Studio 太智能了，已经帮我们把一些简单内容都自动生成了。你也别心急，后面写代码的机会多着呢，我们先来分析一下 HelloWorld 这个项目吧。

1.3.4 分析你的第一个 Android 程序

回到 Android Studio 当中，首先展开 HelloWorld 项目，你会看到如图 1.26 所示的项目结构。

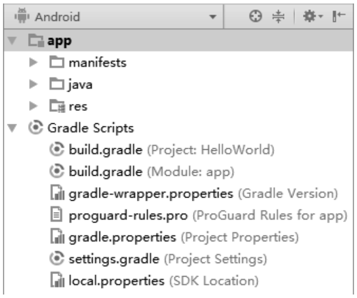


图 1.26　Android 模式的项目结构

任何一个新建的项目都会默认使用 Android 模式的项目结构，但这并不是项目真实的目录结构，而是被 Android Studio 转换过的。这种项目结构简洁明了，适合进行快速开发，但是对于新手来说可能并不易于理解。点击图 1.26 当中的 Android 区域可以切换项目结构模式，如图 1.27 所示。

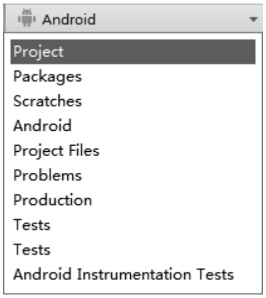


图 1.27　切换项目结构模式

这里我们将项目结构模式切换成 Project，这就是项目真实的目录结构了，如图 1.28 所示。

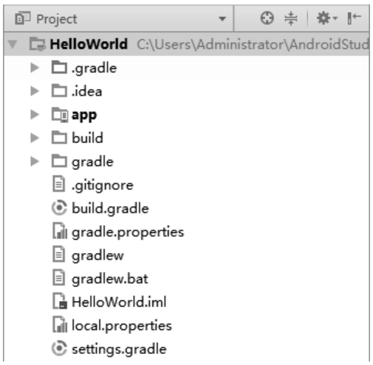


图 1.28　Project 模式的项目结构

一开始看到这么多陌生的东西，你一定会感到有点头晕吧。别担心，我现在就对图 1.28 中的内容进行一一讲解，之后你再看这张图就不会感到那么吃力了。

1. .gradle 和.idea

这两个目录下放置的都是 Android Studio 自动生成的一些文件，我们无须关心，也不要去手动编辑。

2. app

项目中的代码、资源等内容几乎都是放置在这个目录下的，我们后面的开发工作也基本都是在这个目录下进行的，待会儿还会对这个目录单独展开进行讲解。

3. build

这个目录你也不需要过多关心，它主要包含了一些在编译时自动生成的文件。

4. gradle

这个目录下包含了 gradle wrapper 的配置文件，使用 gradle wrapper 的方式不需要提前将 gradle 下载好，而是会自动根据本地的缓存情况决定是否需要联网下载 gradle。Android Studio 默认没有启用 gradle wrapper 的方式，如果需要打开，可以点击 Android Studio 导航栏→File→Settings→Build, Execution, Deployment→Gradle，进行配置更改。

5. .gitignore

这个文件是用来将指定的目录或文件排除在版本控制之外的，关于版本控制我们将在第 5 章中开始正式的学习。

6. build.gradle

这是项目全局的 gradle 构建脚本，通常这个文件中的内容是不需要修改的。稍后我们将会详细分析 gradle 构建脚本中的具体内容。

7. gradle.properties

这个文件是全局的 gradle 配置文件，在这里配置的属性将会影响到项目中所有的 gradle 编译脚本。

8. gradlew 和 gradlew.bat

这两个文件是用来在命令行界面中执行 gradle 命令的，其中 gradlew 是在 Linux 或 Mac 系统中使用的，gradlew.bat 是在 Windows 系统中使用的。

9. HelloWorld.iml

iml 文件是所有 IntelliJ IDEA 项目都会自动生成的一个文件（Android Studio 是基于 IntelliJ IDEA 开发的），用于标识这是一个 IntelliJ IDEA 项目，我们不需要修改这个文件中的任何内容。

10. local.properties

这个文件用于指定本机中的 Android SDK 路径，通常内容都是自动生成的，我们并不需要修改。除非你本机中的 Android SDK 位置发生了变化，那么就将这个文件中的路径改成新的位置即可。

11. settings.gradle

这个文件用于指定项目中所有引入的模块。由于 HelloWorld 项目中就只有一个 app 模块，因此该文件中也就只引入了 app 这一个模块。通常情况下模块的引入都是自动完成的，需要我们手动去修改这个文件的场景可能比较少。

现在整个项目的外层目录结构已经介绍完了。你会发现，除了 app 目录之外，大多数的文件和目录都是自动生成的，我们并不需要进行修改。想必你已经猜到了，app 目录下的内容才是我们以后的工作重点，展开之后结构如图 1.29 所示。

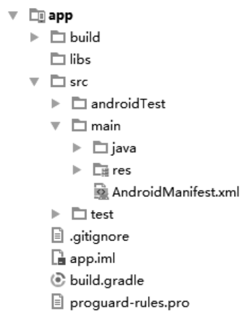


图 1.29　app 目录下的结构

那么下面我们就来对 app 目录下的内容进行更为详细的分析。

1. build

这个目录和外层的 build 目录类似，主要也是包含了一些在编译时自动生成的文件，不过它里面的内容会更多更杂，我们不需要过多关心。

2. libs

如果你的项目中使用到了第三方 jar 包，就需要把这些 jar 包都放在 libs 目录下，放在这个目录下的 jar 包都会被自动添加到构建路径里去。

3. androidTest

此处是用来编写 Android Test 测试用例的，可以对项目进行一些自动化测试。

4. java

毫无疑问，java 目录是放置我们所有 Java 代码的地方，展开该目录，你将看到我们刚才创建的 HelloWorldActivity 文件就在里面。

5. res

这个目录下的内容就有点多了。简单点说，就是你在项目中使用到的所有图片、布局、字符串等资源都要存放在这个目录下。当然这个目录下还有很多子目录，图片放在 drawable 目录下，布局放在 layout 目录下，字符串放在 values 目录下，所以你不用担心会把整个 res 目录弄得乱糟糟的。

6. AndroidManifest.xml

这是你整个 Android 项目的配置文件，你在程序中定义的所有四大组件都需要在这个文件里注册，另外还可以在这个文件中给应用程序添加权限声明。由于这个文件以后会经常用到，我们用到的时候再做详细说明。

7. test

此处是用来编写 Unit Test 测试用例的，是对项目进行自动化测试的另一种方式。

8. .gitignore

这个文件用于将 app 模块内的指定的目录或文件排除在版本控制之外，作用和外层的.gitignore 文件类似。

9. app.iml

IntelliJ IDEA 项目自动生成的文件，我们不需要关心或修改这个文件中的内容。

10. build.gradle

这是 app 模块的 gradle 构建脚本，这个文件中会指定很多项目构建相关的配置，我们稍后将会详细分析 gradle 构建脚本中的具体内容。

11. proguard-rules.pro

这个文件用于指定项目代码的混淆规则，当代码开发完成后打成安装包文件，如果不希望代码被别人破解，通常会将代码进行混淆，从而让破解者难以阅读。

这样整个项目的目录结构就都介绍完了，如果你还不能完全理解的话也很正常，毕竟里面有太多的东西你都还没接触过。不过不用担心，这并不会影响到你后面的学习。等你学完整本书再回来看这个目录结构图时，你会觉得特别地清晰和简单。

接下来我们一起分析一下 HelloWorld 项目究竟是怎么运行起来的吧。首先打开 AndroidManifest.xml 文件，从中可以找到如下代码：

```
<activity android:name=".HelloWorldActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

这段代码表示对 HelloWorldActivity 这个活动进行注册，没有在 AndroidManifest.xml 里注册的活动是不能使用的。其中 `intent-filter` 里的两行代码非常重要，`<action android:name="android.intent.action.MAIN" />`和`<category android:name="android.intent.category.LAUNCHER" />`表示 HelloWorldActivity 是这个项目的主活动，在手机上点击应用图标，首先启动的就是这个活动。

那 HelloWorldActivity 具体又有什么作用呢？我在介绍 Android 四大组件的时候说过，活动

是 Android 应用程序的门面，凡是在应用中你看得到的东西，都是放在活动中的。因此你在图 1.24 中看到的界面，其实就是 HelloWorldActivity 这个活动。那我们快去看一下它的代码吧，打开 HelloWorldActivity，代码如下所示：

```
public class HelloWorldActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.hello_world_layout);
    }

}
```

首先我们可以看到，HelloWorldActivity 是继承自 AppCompatActivity 的，这是一种向下兼容的 Activity，可以将 Activity 在各个系统版本中增加的特性和功能最低兼容到 Android 2.1 系统。Activity 是 Android 系统提供的一个活动基类，我们项目中所有的活动都必须继承它或者它的子类才能拥有活动的特性（`AppCompatActivity` 是 Activity 的子类）。然后可以看到 HelloWorldActivity 中有一个 `onCreate()` 方法，这个方法是一个活动被创建时必定要执行的方法，其中只有两行代码，并且没有 Hello World!的字样。那么图 1.24 中显示的 Hello World!是在哪里定义的呢？

其实 Android 程序的设计讲究逻辑和视图分离，因此是不推荐在活动中直接编写界面的，更加通用的一种做法是，在布局文件中编写界面，然后在活动中引入进来。可以看到，在 `onCreate()` 方法的第二行调用了 `setContentView()` 方法，就是这个方法给当前的活动引入了一个 hello_world_layout 布局，那 Hello World!一定就是在这里定义的了！我们快打开这个文件看一看。

布局文件都是定义在 res/layout 目录下的，当你展开 layout 目录，你会看到 hello_world_layout.xml 这个文件。打开该文件并切换到 Text 视图，代码如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/hello_world_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.helloworld.HelloWorldActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

现在还看不懂？没关系，后面我会对布局进行详细讲解的，你现在只需要看到上面代码中有

一个 TextView，这是 Android 系统提供的一个控件，用于在布局中显示文字的。然后你终于在 TextView 中看到了 Hello World!的字样！哈哈！终于找到了，原来就是通过 `android:text="Hello World!"`这句代码定义的。

这样我们就将 HelloWorld 项目的目录结构以及基本的执行过程都分析完了，相信你对 Android 项目已经有了一个初步的认识，下一小节中我们就来学习一下项目中所包含的资源。

1.3.5　详解项目中的资源

如果你展开 res 目录看一下，其实里面的东西还是挺多的，很容易让人看得眼花缭乱，如图 1.30 所示。

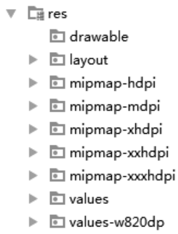


图 1.30　res 目录下的结构

看到这么多的文件夹也不用害怕，其实归纳一下，res 目录就变得非常简单了。所有以 drawable 开头的文件夹都是用来放图片的，所有以 mipmap 开头的文件夹都是用来放应用图标的，所有以 values 开头的文件夹都是用来放字符串、样式、颜色等配置的，layout 文件夹是用来放布局文件的。怎么样，是不是突然感觉清晰了很多?

之所以有这么多 mipmap 开头的文件夹，其实主要是为了让程序能够更好地兼容各种设备。drawable 文件夹也是相同的道理，虽然 Android Studio 没有帮我们自动生成，但是我们应该自己创建 drawable-hdpi、drawable-xhdpi、drawable-xxhdpi 等文件夹。在制作程序的时候最好能够给同一张图片提供几个不同分辨率的版本，分别放在这些文件夹下，然后当程序运行的时候，会自动根据当前运行设备分辨率的高低选择加载哪个文件夹下的图片。当然这只是理想情况，更多的时候美工只会提供给我们一份图片，这时你就把所有图片都放在 drawable-xxhdpi 文件夹下就好了。

知道了 res 目录下每个文件夹的含义，我们再来看一下如何去使用这些资源吧。打开 res/values/strings.xml 文件，内容如下所示：

```
<resources>
    <string name="app_name">HelloWorld</string>
</resources>
```

可以看到，这里定义了一个应用程序名的字符串，我们有以下两种方式来引用它。

- 在代码中通过 `R.string.app_name` 可以获得该字符串的引用。
- 在 XML 中通过`@string/app_name` 可以获得该字符串的引用。

基本的语法就是上面这两种方式，其中 `string` 部分是可以替换的，如果是引用的图片资源就可以替换成 `drawable`，如果是引用的应用图标就可以替换成 `mipmap`，如果是引用的布局文件就可以替换成 `layout`，以此类推。

下面举一个简单的例子来帮助你理解，打开 AndroidManifest.xml 文件，找到如下代码：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    ...
</application>
```

其中，HelloWorld 项目的应用图标就是通过 `android:icon` 属性来指定的，应用的名称则是通过 `android:label` 属性指定的。可以看到，这里对资源引用的方式正是我们刚刚学过的在 XML 中引用资源的语法。

经过本小节的学习，如果你想修改应用的图标或者名称，相信已经知道该怎么办了吧。

1.3.6 详解 build.gradle 文件

不同于 Eclipse，Android Studio 是采用 Gradle 来构建项目的。Gradle 是一个非常先进的项目构建工具，它使用了一种基于 Groovy 的领域特定语言（DSL）来声明项目设置，摒弃了传统基于 XML（如 Ant 和 Maven）的各种烦琐配置。

在 1.3.4 小节中我们已经看到，HelloWorld 项目中有两个 build.gradle 文件，一个是在最外层目录下的，一个是在 app 目录下的。这两个文件对构建 Android Studio 项目都起到了至关重要的作用，下面我们就来对这两个文件中的内容进行详细的分析。

先来看一下最外层目录下的 build.gradle 文件，代码如下所示：

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.0'
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

这些代码都是自动生成的，虽然语法结构看上去可能有点难以理解，但是如果我们忽略语法结构，只看最关键的部分，其实还是很好懂的。

首先，两处 `repositories` 的闭包中都声明了 `jcenter()`这行配置，那么这个 jcenter 是什么意思呢？其实它是一个代码托管仓库，很多 Android 开源项目都会选择将代码托管到 jcenter 上，声明了这行配置之后，我们就可以在项目中轻松引用任何 jcenter 上的开源项目了。

接下来，`dependencies` 闭包中使用 `classpath` 声明了一个 Gradle 插件。为什么要声明这个插件呢？因为 Gradle 并不是专门为构建 Android 项目而开发的，Java、C++等很多种项目都可以使用 Gradle 来构建。因此如果我们要想使用它来构建 Android 项目，则需要声明 `com.android.tools.build:gradle:2.2.0` 这个插件。其中，最后面的部分是插件的版本号，我在写作本书时最新的插件版本是 2.2.0。

这样我们就将最外层目录下的 build.gradle 文件分析完了，通常情况下你并不需要修改这个文件中的内容，除非你想添加一些全局的项目构建配置。

下面我们再来看一下 app 目录下的 build.gradle 文件，代码如下所示：

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.example.helloworld"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

这个文件中的内容就要相对复杂一些了，下面我们一行行地进行分析。首先第一行应用了一个插件，一般有两种值可选：`com.android.application` 表示这是一个应用程序模块，`com.android.library` 表示这是一个库模块。应用程序模块和库模块的最大区别在于，一个是可以直接运行的，一个只能作为代码库依附于别的应用程序模块来运行。

接下来是一个大的 android 闭包，在这个闭包中我们可以配置项目构建的各种属性。其中，`compileSdkVersion` 用于指定项目的编译版本，这里指定成 24 表示使用 Android 7.0 系统的 SDK 编译。`buildToolsVersion` 用于指定项目构建工具的版本，目前最新的版本就是 24.0.2，如果

有更新的版本时，Android Studio 会进行提示。

然后我们看到，这里在 android 闭包中又嵌套了一个 **defaultConfig 闭包**，defaultConfig 闭包中可以对项目的更多细节进行配置。其中，`applicationId` 用于指定项目的包名，前面我们在创建项目的时候其实已经指定过包名了，如果你想在后面对其进行修改，那么就是在这里修改的。`minSdkVersion` 用于指定项目最低兼容的 Android 系统版本，这里指定成 15 表示最低兼容到 Android 4.0 系统。`targetSdkVersion` 指定的值表示你在该目标版本上已经做过了充分的测试，系统将会为你的应用程序启用一些最新的功能和特性。比如说 Android 6.0 系统中引入了运行时权限这个功能，如果你将 `targetSdkVersion` 指定成 23 或者更高，那么系统就会为你的程序启用运行时权限功能，而如果你将 `targetSdkVersion` 指定成 22，那么就说明你的程序最高只在 Android 5.1 系统上做过充分的测试，Android 6.0 系统中引入的新功能自然就不会启用了。剩下的两个属性都比较简单，`versionCode` 用于指定项目的版本号，`versionName` 用于指定项目的版本名，这两个属性在生成安装文件的时候非常重要，我们在后面都会学到。

分析完了 defaultConfig 闭包，接下来我们看一下 **buildTypes 闭包**。buildTypes 闭包中用于指定生成安装文件的相关配置，通常只会有两个子闭包，一个是 debug，一个是 release。debug 闭包用于指定生成测试版安装文件的配置，release 闭包用于指定生成正式版安装文件的配置。另外，debug 闭包是可以忽略不写的，因此我们看到上面的代码中就只有一个 release 闭包。下面来看一下 release 闭包中的具体内容吧，`minifyEnabled` 用于指定是否对项目的代码进行混淆，`true` 表示混淆，`false` 表示不混淆。`proguardFiles` 用于指定混淆时使用的规则文件，这里指定了两个文件，第一个 `proguard-android.txt` 是在 Android SDK 目录下的，里面是所有项目通用的混淆规则，第二个 `proguard-rules.pro` 是在当前项目的根目录下的，里面可以编写当前项目特有的混淆规则。需要注意的是，通过 Android Studio 直接运行项目生成的都是测试版安装文件，关于如何生成正式版安装文件我们将会在第 15 章中学习。

这样整个 android 闭包中的内容就都分析完了，接下来还剩一个 **dependencies 闭包**。这个闭包的功能非常强大，它可以指定当前项目所有的依赖关系。通常 Android Studio 项目一共有 3 种依赖方式：本地依赖、库依赖和远程依赖。本地依赖可以对本地的 Jar 包或目录添加依赖关系，库依赖可以对项目中的库模块添加依赖关系，远程依赖则可以对 jcenter 库上的开源项目添加依赖关系。观察一下 dependencies 闭包中的配置，第一行的 `compile fileTree` 就是一个本地依赖声明，它表示将 libs 目录下所有.jar 后缀的文件都添加到项目的构建路径当中。而第二行的 `compile` 则是远程依赖声明，`com.android.support:appcompat-v7:24.2.1`就是一个标准的远程依赖库格式，其中 `com.android.support` 是域名部分，用于和其他公司的库做区分；`appcompat-v7` 是组名称，用于和同一个公司中不同的库做区分；24.2.1 是版本号，用于和同一个库不同的版本做区分。加上这句声明后，Gradle 在构建项目时会首先检查一下本地是否已经有这个库的缓存，如果没有的话则会去自动联网下载，然后再添加到项目的构建路径当中。至于库依赖声明这里没有用到，它的基本格式是 `compile project` 后面加上要依赖的库名称，比如说有一个库模块的名字叫 helper，那么添加这个库的依赖关系只需要加入 `compile project(':helper')`这句声明即可。另外剩下

的一句 `testCompile` 是用于声明测试用例库的，这个我们暂时用不到，先忽略它就可以了。

1.4　前行必备——掌握日志工具的使用

通过上一节的学习，你已经成功创建了你的第一个 Android 程序，并且对 Android 项目的目录结构和运行流程都有了一定的了解。现在本应该是你继续前行的时候，不过我想在这里给你穿插一点内容，讲解一下 Android 中日志工具的使用方法，这对你以后的 Android 开发之旅会有极大的帮助。

1.4.1　使用 Android 的日志工具 Log

Android 中的日志工具类是 Log（android.util.Log），这个类中提供了如下 5 个方法来供我们打印日志。

- `Log.v()`。用于打印那些最为琐碎的、意义最小的日志信息。对应级别 verbose，是 Android 日志里面级别最低的一种。
- `Log.d()`。用于打印一些调试信息，这些信息对你调试程序和分析问题应该是有帮助的。对应级别 debug，比 verbose 高一级。
- `Log.i()`。用于打印一些比较重要的数据，这些数据应该是你非常想看到的、可以帮你分析用户行为数据。对应级别 info，比 debug 高一级。
- `Log.w()`。用于打印一些警告信息，提示程序在这个地方可能会有潜在的风险，最好去修复一下这些出现警告的地方。对应级别 warn，比 info 高一级。
- `Log.e()`。用于打印程序中的错误信息，比如程序进入到了 catch 语句当中。当有错误信息打印出来的时候，一般都代表你的程序出现严重问题了，必须尽快修复。对应级别 error，比 warn 高一级。

其实很简单，一共就 5 个方法，当然每个方法还会有不同的重载，但那对你来说肯定不是什么难理解的地方了。我们现在就在 HelloWorld 项目中试一试日志工具好不好用吧。

打开 HelloWorldActivity，在 `onCreate()`方法中添加一行打印日志的语句，如下所示：

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.hello_world_layout);
    Log.d("HelloWorldActivity", "onCreate execute");
}
```

`Log.d()`方法中传入了两个参数：第一个参数是 `tag`，一般传入当前的类名就好，主要用于对打印信息进行过滤；第二个参数是 `msg`，即想要打印的具体的内容。

现在可以重新运行一下 HelloWorld 这个项目了，点击顶部工具栏上的运行按钮，或者使用快捷键 Shift + F10（Mac 系统是 control + R），等程序运行完毕，点击 Android Studio 底部工具栏的 Android Monitor，在 logcat 中就可以看到打印信息了，如图 1.31 所示。

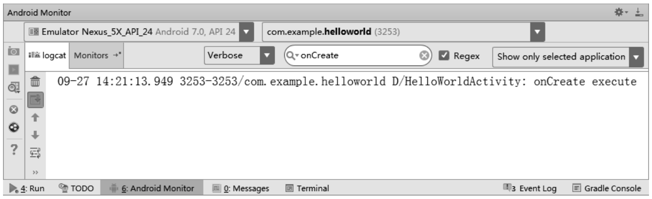


图 1.31　logcat 中的打印信息

其中，你不仅可以看到打印日志的内容和 tag 名，就连程序的包名、打印的时间以及应用程序的进程号都可以看到。

另外，不知道你有没有注意到，你的第一行代码已经在不知不觉中写出来了，我也总算是交差了。

1.4.2　为什么使用 Log 而不使用 System.out

我相信很多的 Java 新手都非常喜欢使用 `System.out.println()`方法来打印日志，不知道你是不是也喜欢这么做。不过在真正的项目开发中，是极度不建议使用 `System.out.println()`方法的！如果你在公司的项目中经常使用这个方法，就很有可能要挨骂了。

为什么 `System.out.println()`方法会这么遭大家唾弃呢？经过我仔细分析之后，发现这个方法除了使用方便一点之外，其他就一无是处了。方便在哪儿呢？在 Eclipse 中你只需要输入 syso，然后按下代码提示键，这个方法就会自动出来了，相信这也是很多 Java 新手对它钟情的原因。那缺点又在哪儿了呢？这个就太多了，比如日志打印不可控制、打印时间无法确定、不能添加过滤器、日志没有级别区分……

听我说了这些，你可能已经不太想用 `System.out.println()`方法了，那么 Log 就把上面所说的缺点全部都改好了吗？虽然谈不上全部，但我觉得 Log 已经做得相当不错了。我现在就来带你看看 Log 和 logcat 配合的强大之处。

首先刚才提到的快捷输入，在 Android Studio 当中也是有的，比如你想打印一条 debug 级别的日志，那么只需要输入 logd，然后按下 Tab 键，就会帮你自动补全一条完整的打印语句。输入 logi，然后按下 Tab 键，会自动补全一条 info 级别的打印日志。输入 logw，按下 Tab 键，会自动补全一条 warn 级别的打印日志，以此类推。另外，由于 Log 的所有打印方法都要求传入一个 tag 参数，每次写一遍显然太过麻烦。这里还有一个小技巧，我们在 `onCreate()`方法的外面输入 logt，然后按下 Tab 键，这时就会以当前的类名作为值自动生成一个 TAG 常量，如下所示：

```
public class HelloWorldActivity extends AppCompatActivity {

    private static final String TAG = "HelloWorldActivity";

    ...
}
```

除了快捷输入之外，logcat 中还能很轻松地添加过滤器，你可以在图 1.32 中看到我们目前所有的过滤器。

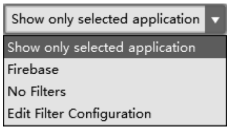


图 1.32　logcat 中的过滤器

目前只有 3 个过滤器，Show only selected application 表示只显示当前选中程序的日志，Firebase 是谷歌提供的一个分析工具，我们可以不用管它，No Filters 相当于没有过滤器，会把所有的日志都显示出来。那可不可以自定义过滤器呢？当然可以，我们现在就来添加一个过滤器试试。

点击图 1.32 中的 Edit Filter Configuration，会弹出一个过滤器配置界面。我们给过滤器起名叫 data，并且让它对名为 data 的 tag 进行过滤，如图 1.33 所示。

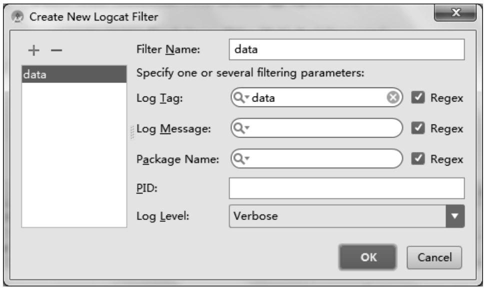


图 1.33　过滤器配置界面

点击 OK，你就会发现你已经多出了一个 data 过滤器。当你点击这个过滤器的时候，你会发现刚才在 `onCreate()`方法里打印的日志没了，这是因为 data 这个过滤器只会显示 tag 名称为 data 的日志。你可以尝试在 `onCreate()`方法中把打印日志的语句改成 `Log.d("data", "onCreate execute")`，然后再次运行程序，你就会在 data 过滤器下看到这行日志了。

不知道你有没有体会到使用过滤器的好处，可能现在还没有吧。不过当你的程序打印出成百上千行日志的时候，你就会迫切地需要过滤器了。

看完了过滤器，再来看一下 logcat 中的日志级别控制吧。logcat 中主要有 5 个级别，分别对应着上一节介绍的 5 个方法，如图 1.34 所示。



图 1.34　logcat 中的日志级别

当前我们选中的级别是 verbose，也就是最低等级。这意味着不管我们使用哪一个方法打印日志，这条日志都一定会显示出来。而如果我们将级别选中为 debug，这时只有我们使用 debug 及以上级别方法打印的日志才会显示出来，以此类推。你可以做一下试验，当你把 logcat 中的级别选中为 info、warn 或者 error 时，我们在 `onCreate()`方法中打印的语句是不会显示的，因为我们打印日志时使用的是 `Log.d()`方法。

日志级别控制的好处就是，你可以很快地找到你所关心的那些日志。相信如果让你从上千行日志中查找一条崩溃信息，你一定会抓狂的吧。而现在你只需要将日志级别选中为 error，那些不相干的琐碎信息就不会再干扰你的视线了。

最后我们再来看一下关键字过滤。如果使用过滤器加日志级别控制还是不能锁定到你想查看的日志内容的话，那么还可以通过关键字进行进一步的过滤，如图 1.35 所示。



图 1.35 关键字输入框

我们可以在输入框里输入关键字的内容，这样只有符合关键字条件的日志才会显示出来，从而能够快速定位到任何你想查看的日志。另外还有一点需要注意，关键字过滤是支持正则表达式的，有了这个特性，我们就可以构建出更加丰富的过滤条件。

关于 Android 中日志工具的使用我就准备讲到这里，logcat 中其他的一些使用技巧就要靠你自己去摸索了。今天你已经学到了足够多的东西，我们来总结和梳理一下吧。

1.5 小结与点评

你现在一定会觉得很充实，甚至有点沾沾自喜。确实应该如此，因为你已经成为一名真正的 Android 开发者了。通过本章的学习，你首先对 Android 系统有了更加充足的认识，然后成功将 Android 开发环境搭建了起来，接着创建了你自己的第一个 Android 项目，并对 Android 项目的目录结构和执行过程有了一定的认识，在本章的最后还学习了 Android 日志工具的使用，这难道还不够充实吗？

不过你也别太过于满足，相信你很清楚，Android 开发者和出色的 Android 开发者还是有很大的区别的，你还需要付出更多的努力才行。即使你目前在 Java 领域已经有了不错的成绩，我也希望在 Android 的世界你可以放下身段，以一只萌级小菜鸟的身份起飞，在后面的旅途中你会不断地成长。

现在你可以非常安心地休息一段时间，因为今天你已经做得非常不错了。储备好能量，准备进入到下一章的旅程当中。

第 2 章

先从看得到的入手——探究活动

通过上一章的学习，你已经成功创建了你的第一个 Android 项目。不过仅仅满足于此显然是不够的，是时候学点新的东西了。作为你的导师，我有义务帮你制定好后面的学习路线，那么今天我们应该从哪儿入手呢？现在你可以想象一下，假如你已经写出了一个非常优秀的应用程序，然后推荐给你的第一个用户，你会从哪里开始介绍呢？毫无疑问，当然是从界面开始介绍了！因为即使你的程序算法再高效，架构再出色，用户根本不会在乎这些，他们一开始只会对看得到的东西感兴趣，那么我们今天的主题自然也要从看得到的入手了。

2.1 活动是什么

活动（Activity）是最容易吸引用户的地方，它是一种可以包含用户界面的组件，主要用于和用户进行交互。一个应用程序中可以包含零个或多个活动，但不包含任何活动的应用程序很少见，谁也不想让自己的应用永远无法被用户看到吧？

其实在上一章中，你已经和活动打过交道了，并且对活动也有了初步的认识。不过上一章我们的重点是创建你的第一个 Android 项目，对活动的介绍并不多，在本章中我将对活动进行详细的介绍。

2.2 活动的基本用法

到现在为止，你还没有手动创建过活动呢，因为上一章中的 HelloWorldActivity 是 Android Studio 帮我们自动创建的。手动创建活动可以加深我们的理解，因此现在是时候应该自己动手了。

由于 Android Studio 在一个工作区间内只允许打开一个项目，因此首先你需要将当前的项目关闭，点击导航栏 File→Close Project。然后再新建一个 Android 项目，项目名可以叫作 ActivityTest，包名我们就使用默认值 com.example.activitytest。新建项目的步骤你已经在上一章学习过了，不过图 1.12 中的那一步需要稍做修改，我们不再选择 Empty Activity 这个选项，而是选择 Add No

Activity，因为这次我们准备手动创建活动，如图 2.1 所示。

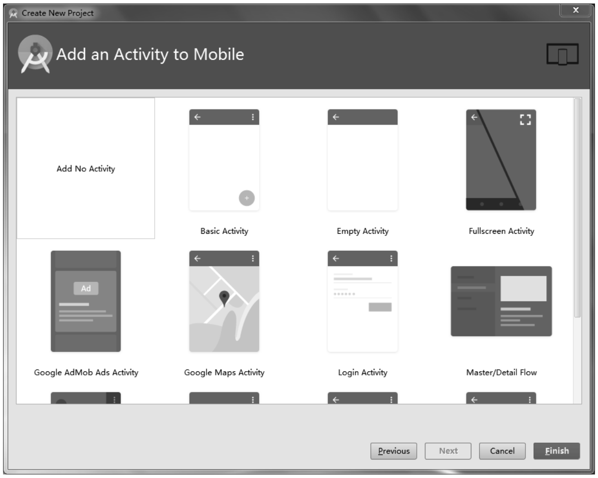


图 2.1 选择不添加活动

点击 Finish，等待 Gradle 构建完成后，项目就创建成功了。

2.2.1 手动创建活动

项目创建成功后，仍然会默认使用 Android 模式的项目结构，这里我们手动改成 Project 模式，本书中后面的所有项目都要这样修改，以后就不再赘述了。目前 ActivityTest 项目中虽然还是会自动生成很多文件，但是 app/src/main/java/com.example.activitytest 目录应该是空的了，如图 2.2 所示。

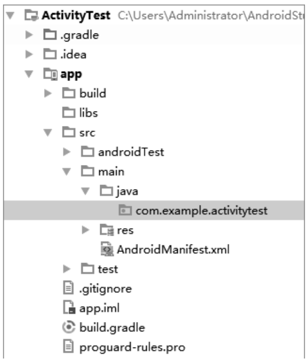


图 2.2 初始项目结构

现在右击 com.example.activitytest 包→New→Activity→Empty Activity，会弹出一个创建活动的对话框，我们将活动命名为 FirstActivity，并且不要勾选 Generate Layout File 和 Launcher Activity 这两个选项，如图 2.3 所示。

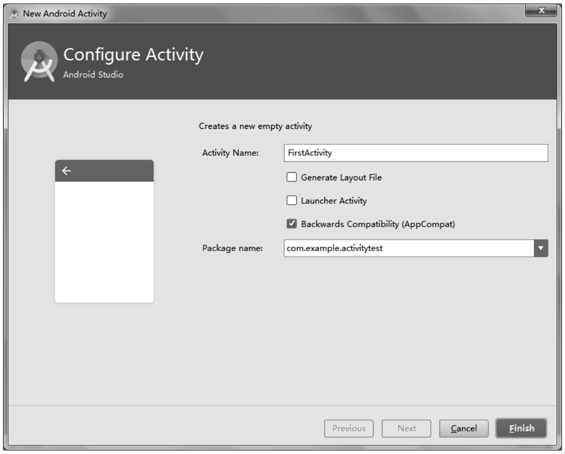


图 2.3　新建活动对话框

勾选 Generate Layout File 表示会自动为 FirstActivity 创建一个对应的布局文件，勾选 Launcher Activity 表示会自动将 FirstActivity 设置为当前项目的主活动，这里由于你是第一次手动创建活动，这些自动生成的东西暂时都不要勾选，下面我们将会一个个手动来完成。勾选 Backwards Compatibility 表示会为项目启用向下兼容的模式，这个选项要勾上。点击 Finish 完成创建。

你需要知道，项目中的任何活动都应该重写 Activity 的 onCreate()方法，而目前我们的 FirstActivity 中已经重写了这个方法，这是由 Android Studio 自动帮我们完成的，代码如下所示：

```
public class FirstActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

}
```

可以看到，onCreate()方法非常简单，就是调用了父类的 onCreate()方法。当然这只是默认的实现，后面我们还需要在里面加入很多自己的逻辑。

2.2.2　创建和加载布局

前面我们说过，Android 程序的设计讲究逻辑和视图分离，最好每一个活动都能对应一个布局，布局就是用来显示界面内容的，因此我们现在就来手动创建一个布局文件。

右击 app/src/main/res 目录→New→Directory，会弹出一个新建目录的窗口，这里先创建一个名为 layout 的目录。然后对着 layout 目录右键→New→Layout resource file，又会弹出一个新建布局资源文件的窗口，我们将这个布局文件命名为 first_layout，根元素就默认选择为 LinearLayout，如图 2.4 所示。

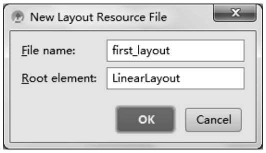


图 2.4　新建布局资源文件

点击 OK 完成布局的创建，这时候你会看到如图 2.5 所示的布局编辑器。

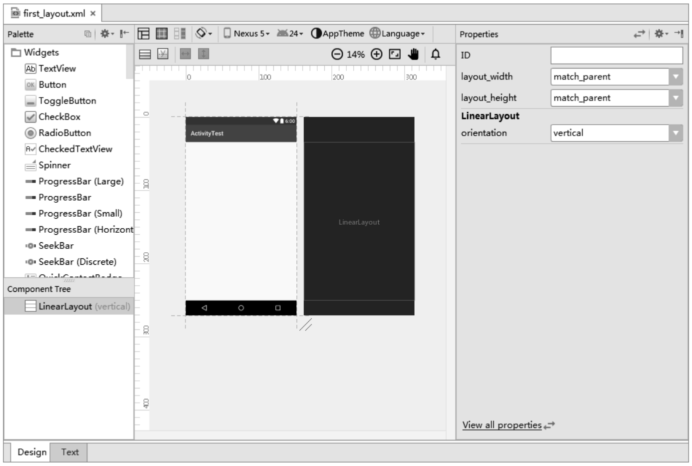


图 2.5　布局编辑器

这是 Android Studio 为我们提供的可视化布局编辑器，你可以在屏幕的中央区域预览当前的布局。在窗口的最下方有两个切换卡，左边是 Design，右边是 Text。Design 是当前的可视化布局编辑器，在这里你不仅可以预览当前的布局，还可以通过拖放的方式编辑布局。而 Text 则是通过 XML 文件的方式来编辑布局的，现在点击一下 Text 切换卡，可以看到如下代码：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</LinearLayout>
```

由于我们刚才在创建布局文件时选择了 LinearLayout 作为根元素，因此现在布局文件中已经有一个 LinearLayout 元素了。那我们现在对这个布局稍做编辑，添加一个按钮，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button_1"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 1"
        />

</LinearLayout>
```

这里添加了一个 Button 元素，并在 Button 元素的内部增加了几个属性。android:id 是给当前的元素定义一个唯一标识符，之后可以在代码中对这个元素进行操作。你可能会对@+id/button_1 这种语法感到陌生，但如果把加号去掉，变成@id/button_1，这样你就会觉得有些熟悉了吧，这不就是在 XML 中引用资源的语法吗？只不过是把 string 替换成了 id。是的，如果你需要在 XML 中引用一个 id，就使用@id/id_name 这种语法，而如果你需要在 XML 中定义一个 id，则要使用@+id/id_name 这种语法。随后 android:layout_width 指定了当前元素的宽度，这里使用 match_parent 表示让当前元素和父元素一样宽。android:layout_height 指定了当前元素的高度，这里使用 wrap_content 表示当前元素的高度只要能刚好包含里面的内容就行。android:text 指定了元素中显示的文字内容。如果你还不能完全看明白，没有关系，关于编写布局的详细内容我会在下一章中重点讲解，本章只是先简单涉及一些。现在按钮已经添加完了，你可以通过右侧工具栏的 Preview 来预览一下当前布局，如图 2.6 所示。

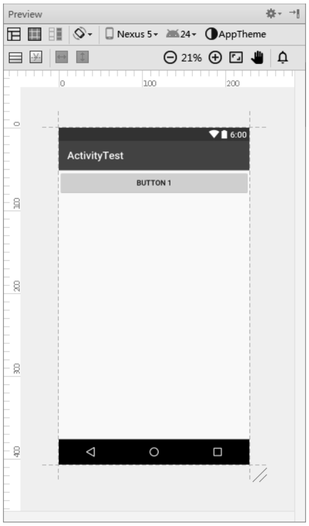


图 2.6　预览当前布局

可以看到，按钮已经成功显示出来了，这样一个简单的布局就编写完成了。那么接下来我们要做的，就是在活动中加载这个布局。

重新回到 FirstActivity，在 onCreate()方法中加入如下代码：

```
public class FirstActivity extends AppCompatActivity {

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_layout);
    }

}
```

可以看到，这里调用了 setContentView()方法来给当前的活动加载一个布局，而在setContentView()方法中，我们一般都会传入一个布局文件的 id。在第 1 章介绍项目资源的时候我曾提到过，项目中添加的任何资源都会在 R 文件中生成一个相应的资源 id，因此我们刚才创建的 first_layout.xml 布局的 id 现在应该是已经添加到 R 文件中了。在代码中去引用布局文件的方法你也已经学过了，只需要调用 R.layout.first_layout 就可以得到 first_layout.xml 布局的 id，然后将这个值传入 setContentView()方法即可。

2.2.3　在 AndroidManifest 文件中注册

别忘了在上一章我们学过，所有的活动都要在 AndroidManifest.xml 中进行注册才能生效，而实际上 FirstActivity 已经在 AndroidManifest.xml 中注册过了，我们打开 app/src/main/AndroidManifest.xml 文件瞧一瞧，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.activitytest">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".FirstActivity"></activity>
    </application>
</manifest>
```

可以看到，活动的注册声明要放在<application>标签内，这里是通过<activity>标签来对活动进行注册的。那么又是谁帮我们自动完成了对 FirstActivity 的注册呢？当然是 Android Studio 了，之前在使用 Eclipse 创建活动或其他系统组件时，很多人都会忘记要去 Android Manifest.xml 中注册一下，从而导致程序运行崩溃，很显然 Android Studio 在这方面做得更加人性化。

在<activity>标签中我们使用了 android:name 来指定具体注册哪一个活动，那么这里填入的.FirstActivity 是什么意思呢？其实这不过就是 com.example.activitytest.FirstActivity 的缩写而已。由于在最外层的<manifest>标签中已经通过 package 属性指定了程序的包名是 com.example.activitytest，因此在注册活动时这一部分就可以省略了，直接使用.FirstActivity 就足够了。

不过，仅仅是这样注册了活动，我们的程序仍然是不能运行的，因为还没有为程序配置主活动，也就是说，当程序运行起来的时候，不知道要首先启动哪个活动。配置主活动的方法其实在第 1 章中已经介绍过了，就是在<activity>标签的内部加入<intent-filter>标签，并在这个

标签里添加<action android:name="android.intent.action.MAIN"/>和<category android:name="android.intent.category.LAUNCHER" />这两句声明即可。

除此之外，我们还可以使用 android:label 指定活动中标题栏的内容，标题栏是显示在活动最顶部的，待会儿运行的时候你就会看到。需要注意的是，给主活动指定的 label 不仅会成为标题栏中的内容，还会成为启动器（Launcher）中应用程序显示的名称。

修改后的 AndroidManifest.xml 文件，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.activitytest">
    <application
        ... >
        <activity android:name=".FirstActivity"
            android:label="This is FirstActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

这样的话，FirstActivity 就成为我们这个程序的主活动了，即点击桌面应用程序图标时首先打开的就是这个活动。另外需要注意，如果你的应用程序中没有声明任何一个活动作为主活动，这个程序仍然是可以正常安装的，只是你无法在启动器中看到或者打开这个程序。这种程序一般都是作为第三方服务供其他应用在内部进行调用的，如支付宝快捷支付服务。

好了，现在一切都已准备就绪，让我们来运行一下程序吧，结果如图 2.7 所示。

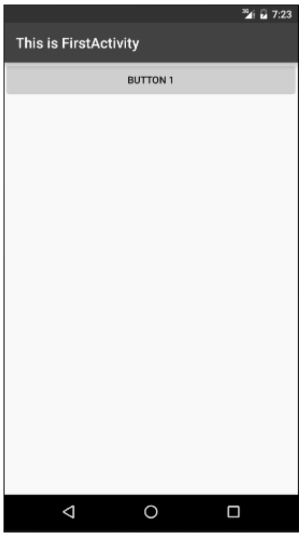


图 2.7　首次运行结果

在界面的最顶部是一个标题栏，里面显示着我们刚才在注册活动时指定的内容。标题栏的下面就是在布局文件 first_layout.xml 中编写的界面，可以看到我们刚刚定义的按钮。现在你已经成功掌握了手动创建活动的方法，下面让我们继续看一看你在活动中还能做哪些事情吧。

2.2.4　在活动中使用 Toast

Toast 是 Android 系统提供的一种非常好的提醒方式，在程序中可以使用它将一些短小的信息通知给用户，这些信息会在一段时间后自动消失，并且不会占用任何屏幕空间，我们现在就尝试一下如何在活动中使用 Toast。

首先需要定义一个弹出 Toast 的触发点，正好界面上有个按钮，那我们就让点击这个按钮的时候弹出一个 Toast 吧。在 `onCreate()`方法中添加如下代码：

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.first_layout);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(FirstActivity.this, "You clicked Button 1",
                Toast.LENGTH_SHORT).show();
        }
    });
}
```

在活动中，可以通过 `findViewById()`方法获取到在布局文件中定义的元素，这里我们传入 `R.id.button_1`，来得到按钮的实例，这个值是刚才在 first_layout.xml 中通过 `android:id` 属性指定的。`findViewById()`方法返回的是一个 `View` 对象，我们需要向下转型将它转成 `Button` 对象。得到按钮的实例之后，我们通过调用 `setOnClickListener()`方法为按钮注册一个监听器，点击按钮时就会执行监听器中的 `onClick()`方法。因此，弹出 Toast 的功能当然是要在 `onClick()`方法中编写了。

Toast 的用法非常简单，通过静态方法 `makeText()`创建出一个 `Toast` 对象，然后调用 `show()`将 Toast 显示出来就可以了。这里需要注意的是，`makeText()`方法需要传入 3 个参数。第一个参数是 `Context`，也就是 Toast 要求的上下文，由于活动本身就是一个 `Context` 对象，因此这里直接传入 `FirstActivity.this` 即可。第二个参数是 Toast 显示的文本内容，第三个参数是 Toast 显示的时长，有两个内置常量可以选择 `Toast.LENGTH_SHORT` 和 `Toast.LENGTH_LONG`。

现在重新运行程序，并点击一下按钮，效果如图 2.8 所示。

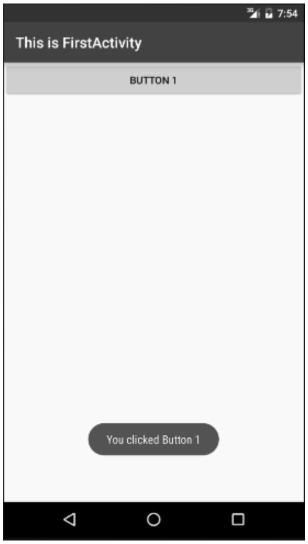


图 2.8　Toast 运行效果

2.2.5　在活动中使用 Menu

手机毕竟和电脑不同，它的屏幕空间非常有限，因此充分地利用屏幕空间在手机界面设计中就显得非常重要了。如果你的活动中有大量的菜单需要显示，这个时候界面设计就会比较尴尬，因为仅这些菜单就可能占用屏幕将近三分之一的空间，这该怎么办呢？不用担心，Android 给我们提供了一种方式，可以让菜单都能得到展示的同时，还能不占用任何屏幕空间。

首先在 res 目录下新建一个 menu 文件夹，右击 res 目录→New→Directory，输入文件夹名 menu，点击 OK。接着在这个文件夹下再新建一个名叫 main 的菜单文件，右击 menu 文件夹→New→Menu resource file，如图 2.9 所示。

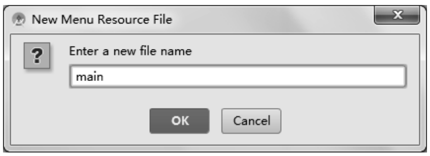


图 2.9　新建 Menu 资源文件

文件名输入 main，点击 OK 完成创建。然后在 main.xml 中添加如下代码：

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/add_item"
        android:title="Add"/>
    <item
        android:id="@+id/remove_item"
```

```
        android:title="Remove"/>
</menu>
```

这里我们创建了两个菜单项，其中<item>标签就是用来创建具体的某一个菜单项，然后通过 android:id 给这个菜单项指定一个唯一的标识符，通过 android:title 给这个菜单项指定一个名称。

接着重新回到 FirstActivity 中来重写 onCreateOptionsMenu()方法，重写方法可以使用 Ctrl + O 快捷键（Mac 系统是 control + O），如图 2.10 所示。

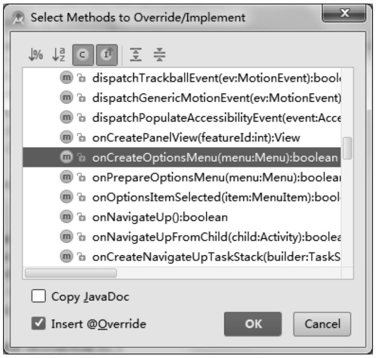


图 2.10　重写 onCreateOptionsMenu()方法

然后在 onCreateOptionsMenu()方法中编写如下代码：

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```

通过 getMenuInflater()方法能够得到 MenuInflater 对象，再调用它的 inflate()方法就可以给当前活动创建菜单了。inflate()方法接收两个参数，第一个参数用于指定我们通过哪一个资源文件来创建菜单，这里当然传入 R.menu.main。第二个参数用于指定我们的菜单项将添加到哪一个 Menu 对象当中，这里直接使用 onCreateOptionsMenu()方法中传入的 menu 参数。然后给这个方法返回 true，表示允许创建的菜单显示出来，如果返回了 false，创建的菜单将无法显示。

当然，仅仅让菜单显示出来是不够的，我们定义菜单不仅是为了看的，关键是要菜单真正可用才行，因此还要再定义菜单响应事件。在 FirstActivity 中重写 onOptionsItemSelected()方法：

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.add_item:
        Toast.makeText(this, "You clicked Add", Toast.LENGTH_SHORT).show();
        break;
    case R.id.remove_item:
        Toast.makeText(this, "You clicked Remove", Toast.LENGTH_SHORT).show();
```

```
            break;
        default:
        }
        return true;
    }
```

在 onOptionsItemSelected()方法中，通过调用 item.getItemId()来判断我们点击的是哪一个菜单项，然后给每个菜单项加入自己的逻辑处理，这里我们就活学活用，弹出一个刚刚学会的 Toast。

重新运行程序，你会发现在标题栏的右侧多了一个三点的符号，这个就是菜单按钮了，如图 2.11 所示。

可以看到，菜单里的菜单项默认是不会显示出来的，只有点击一下菜单按钮才会弹出里面具体的内容，因此它不会占用任何活动的空间，如图 2.12 所示。

然后如果你点击了 Add 菜单项就会弹出 You clicked Add 提示（如图 2.13 所示），如果点击了 Remove 菜单项就会弹出 You clicked Remove 提示。

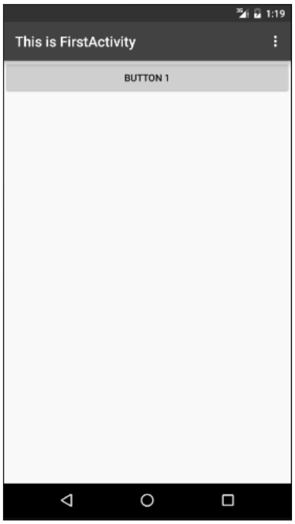


图 2.11　带菜单按钮的活动

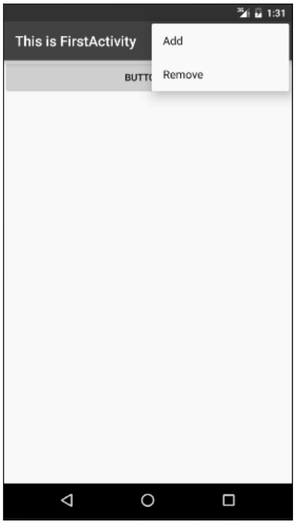


图 2.12　弹出菜单项的界面

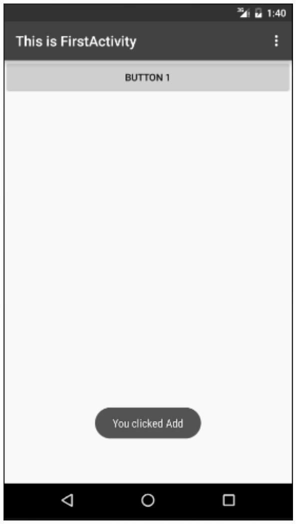


图 2.13　点击了 Add 菜单项

2.2.6　销毁一个活动

通过上一节的学习，你已经掌握了手动创建活动的方法，并学会了如何在活动中创建 Toast 和创建菜单。或许你现在心中会有个疑惑，如何销毁一个活动呢？

其实答案非常简单，只要按一下 Back 键就可以销毁当前的活动了。不过如果你不想通过按键的方式，而是希望在程序中通过代码来销毁活动，当然也可以，Activity 类提供了一个 finish()方法，我们在活动中调用一下这个方法就可以销毁当前活动了。

修改按钮监听器中的代码，如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

重新运行程序，这时点击一下按钮，当前的活动就被成功销毁了，效果和按下 Back 键是一样的。

2.3 使用 Intent 在活动之间穿梭

只有一个活动的应用也太简单了吧？没错，你的追求应该更高一点。不管你想创建多少个活动，方法都和上一节中介绍的是一样的。唯一的问题在于，你在启动器中点击应用的图标只会进入到该应用的主活动，那么怎样才能由主活动跳转到其他活动呢？我们现在就来一起看一看。

2.3.1 使用显式 Intent

你应该已经对创建活动的流程比较熟悉了，那我们现在快速地在 ActivityTest 项目中再创建一个活动。

仍然还是右击 com.example.activitytest 包→New→Activity→Empty Activity，会弹出一个创建活动的对话框，我们这次将活动命名为 SecondActivity，并勾选 Generate Layout File，给布局文件起名为 second_layout，但不要勾选 Launcher Activity 选项，如图 2.14 所示。

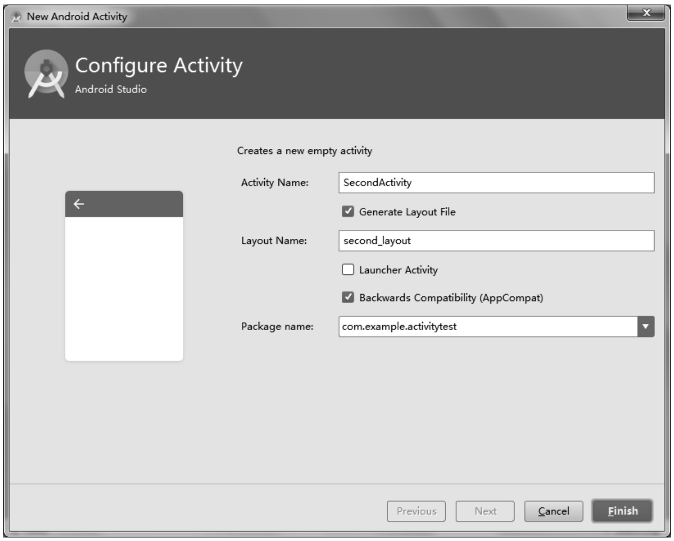


图 2.14 创建 SecondActivity

点击 Finish 完成创建，Android Studio 会为我们自动生成 SecondActivity.java 和 second_layout.xml 这两个文件。不过自动生成的布局代码目前对你来说可能有些复杂，这里我们仍然还是使用最熟悉的 LinearLayout，编辑 second_layout.xml，将里面的代码替换成如下内容：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button_2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 2"
        />

</LinearLayout>
```

我们还是定义了一个按钮，按钮上显示 Button 2。

然后 SecondActivity 中的代码已经自动生成了一部分，我们保持默认不变就好，如下所示：

```
public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second_layout);
    }

}
```

另外不要忘记，任何一个活动都是需要在 AndroidManifest.xml 中注册的，不过幸运的是，Android Studio 已经帮我们自动完成了，你可以打开 AndroidManifest.xml 瞧一瞧：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
        android:name=".FirstActivity"
        android:label="This is FirstActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".SecondActivity"></activity>
</application>
```

由于 SecondActivity 不是主活动，因此不需要配置`<intent-filter>`标签里的内容，注册活动的代码也简单了许多。现在第二个活动已经创建完成，剩下的问题就是如何去启动这第二个活动了，这里我们需要引入一个新的概念：Intent。

Intent 是 Android 程序中各组件之间进行交互的一种重要方式，它不仅可以指明当前组件想要执行的动作，还可以在不同组件之间传递数据。Intent 一般可被用于启动活动、启动服务以及发送广播等场景，由于服务、广播等概念你暂时还未涉及，那么本章我们的目光无疑就锁定在了启动活动上面。

Intent 大致可以分为两种：**显式 Intent** 和**隐式 Intent**，我们先来看一下显式 Intent 如何使用。

Intent 有多个构造函数的重载，其中一个是 `Intent(Context packageContext, Class<?> cls)`。这个构造函数接收两个参数，第一个参数 `Context` 要求提供一个启动活动的上下文，第二个参数 `Class` 则是指定想要启动的目标活动，通过这个构造函数就可以构建出 `Intent` 的“意图”。然后我们应该怎么使用这个 Intent 呢？Activity 类中提供了一个 `startActivity()`方法，这个方法是专门用于启动活动的，它接收一个 `Intent` 参数，这里我们将构建好的 Intent 传入 `startActivity()`方法就可以启动目标活动了。

修改 FirstActivity 中按钮的点击事件，代码如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
        startActivity(intent);
    }
});
```

我们首先构建出了一个 Intent，传入 `FirstActivity.this` 作为上下文，传入 `SecondActivity.class` 作为目标活动，这样我们的“意图”就非常明显了，即在 FirstActivity 这个活动的基础上打开 SecondActivity 这个活动。然后通过 `startActivity()`方法来执行这个 Intent。

重新运行程序，在 FirstActivity 的界面点击一下按钮，结果如图 2.15 所示。

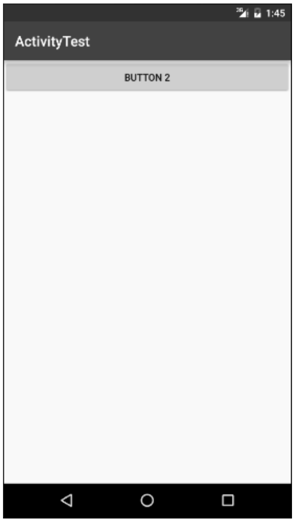


图 2.15　SecondActivity 界面

可以看到，我们已经成功启动 SecondActivity 这个活动了。如果你想要回到上一个活动怎么办呢？很简单，按下 Back 键就可以销毁当前活动，从而回到上一个活动了。

使用这种方式来启动活动，Intent 的“意图”非常明显，因此我们称之为**显式 Intent**。

2.3.2　使用隐式 Intent

相比于显式 Intent，隐式 Intent 则含蓄了许多，它并不明确指出我们想要启动哪一个活动，而是指定了一系列更为抽象的 `action` 和 `category` 等信息，然后交由系统去分析这个 Intent，并帮我们找出合适的活动去启动。

什么叫作合适的活动呢？简单来说就是可以响应我们这个隐式 Intent 的活动，那么目前 SecondActivity 可以响应什么样的隐式 Intent 呢？额，现在好像还什么都响应不了，不过很快就会有了。

通过在`<activity>`标签下配置`<intent-filter>`的内容，可以指定当前活动能够响应的 `action` 和 `category`，打开 AndroidManifest.xml，添加如下代码：

```
<activity android:name=".SecondActivity" >
    <intent-filter>
        <action android:name="com.example.activitytest.ACTION_START" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

在`<action>`标签中我们指明了当前活动可以响应 `com.example.activitytest.ACTION_START` 这个 `action`，而`<category>`标签则包含了一些附加信息，更精确地指明了当前的活动能

够响应的 Intent 中还可能带有的 category。只有<action>和<category>中的内容同时能够匹配上 Intent 中指定的 action 和 category 时，这个活动才能响应该 Intent。

修改 FirstActivity 中按钮的点击事件，代码如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent("com.example.activitytest.ACTION_START");
        startActivity(intent);
    }
});
```

可以看到，我们使用了 Intent 的另一个构造函数，直接将 action 的字符串传了进去，表明我们想要启动能够响应 com.example.activitytest.ACTION_START 这个 action 的活动。那前面不是说要<action>和<category>同时匹配上才能响应的吗？怎么没看到哪里有指定 category 呢？这是因为 android.intent.category.DEFAULT 是一种默认的 category，在调用 startActivity()方法的时候会自动将这个 category 添加到 Intent 中。

重新运行程序，在 FirstActivity 的界面点击一下按钮，你同样成功启动 SecondActivity 了。不同的是，这次你是使用了隐式 Intent 的方式来启动的，说明我们在<activity>标签下配置的 action 和 category 的内容已经生效了！

每个 Intent 中只能指定一个 action，但却能指定多个 category。目前我们的 Intent 中只有一个默认的 category，那么现在再来增加一个吧。

修改 FirstActivity 中按钮的点击事件，代码如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent("com.example.activitytest.ACTION_START");
        intent.addCategory("com.example.activitytest.MY_CATEGORY");
        startActivity(intent);
    }
});
```

可以调用 Intent 中的 addCategory()方法来添加一个 category，这里我们指定了一个自定义的 category，值为 com.example.activitytest.MY_CATEGORY。

现在重新运行程序，在 FirstActivity 的界面点击一下按钮，你会发现，程序崩溃了！这是你第一次遇到程序崩溃，可能会有些束手无策。别紧张，其实大多数的崩溃问题都是很好解决的，只要你善于分析。在 logcat 界面查看错误日志，你会看到如图 2.16 所示的错误信息。

```
Process: com.example.activitytest, PID: 24027
android.content.ActivityNotFoundException: No Activity found to handle Intent {
act=com.example.activitytest.ACTION_START cat=[com.example.activitytest.MY_CATEGORY] }
```

图 2.16 错误信息

错误信息中提醒我们，没有任何一个活动可以响应我们的 Intent，为什么呢？这是因为我们刚刚在 Intent 中新增了一个 `category`，而 SecondActivity 的`<intent-filter>`标签中并没有声明可以响应这个 `category`，所以就出现了没有任何活动可以响应该 Intent 的情况。现在我们在`<intent-filter>`中再添加一个 `category` 的声明，如下所示：

```
<activity android:name=".SecondActivity" >
    <intent-filter>
        <action android:name="com.example.activitytest.ACTION_START" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="com.example.activitytest.MY_CATEGORY"/>
    </intent-filter>
</activity>
```

再次重新运行程序，你就会发现一切都正常了。

2.3.3　更多隐式 Intent 的用法

上一节中，你掌握了通过隐式 Intent 来启动活动的方法，但实际上隐式 Intent 还有更多的内容需要你去了解，本节我们就来展开介绍一下。

使用隐式 Intent，我们不仅可以启动自己程序内的活动，还可以启动其他程序的活动，这使得 Android 多个应用程序之间的功能共享成为了可能。比如说你的应用程序中需要展示一个网页，这时你没有必要自己去实现一个浏览器（事实上也不太可能），而是只需要调用系统的浏览器来打开这个网页就行了。

修改 FirstActivity 中按钮点击事件的代码，如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("http://www.baidu.com"));
        startActivity(intent);
    }
});
```

这里我们首先指定了 Intent 的 `action` 是 `Intent.ACTION_VIEW`，这是一个 Android 系统内置的动作，其常量值为 `android.intent.action.VIEW`。然后通过 `Uri.parse()`方法，将一个网址字符串解析成一个 `Uri` 对象，再调用 Intent 的 `setData()`方法将这个 `Uri` 对象传递进去。

重新运行程序，在 FirstActivity 界面点击按钮就可以看到打开了系统浏览器，如图 2.17 所示。

在上述代码中，可能你会对 `setData()`部分感觉到陌生，这是我们前面没有讲到的。这个方法其实并不复杂，它接收一个 `Uri` 对象，主要用于指定当前 Intent 正在操作的数据，而这些数据通常都是以字符串的形式传入到 `Uri.parse()`方法中解析产生的。



图 2.17　系统浏览器界面

与此对应，我们还可以在<intent-filter>标签中再配置一个<data>标签，用于更精确地指定当前活动能够响应什么类型的数据。<data>标签中主要可以配置以下内容。

- android:scheme。用于指定数据的协议部分，如上例中的 http 部分。
- android:host。用于指定数据的主机名部分，如上例中的 www.baidu.com 部分。
- android:port。用于指定数据的端口部分，一般紧随在主机名之后。
- android:path。用于指定主机名和端口之后的部分，如一段网址中跟在域名之后的内容。
- android:mimeType。用于指定可以处理的数据类型，允许使用通配符的方式进行指定。

只有<data>标签中指定的内容和 Intent 中携带的 Data 完全一致时，当前活动才能够响应该 Intent。不过一般在<data>标签中都不会指定过多的内容，如上面浏览器示例中，其实只需要指定 android:scheme 为 http，就可以响应所有的 http 协议的 Intent 了。

为了让你能够更加直观地理解，我们来自己建立一个活动，让它也能响应打开网页的 Intent。

右击 com.example.activitytest 包→New→Activity→Empty Activity，新建 ThirdActivity，并勾选 Generate Layout File，给布局文件起名为 third_layout，点击 Finish 完成创建。然后编辑 third_layout.xml，将里面的代码替换成如下内容：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
```

```
        android:id="@+id/button_3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 3"
        />

</LinearLayout>
```

ThirdActivity 中的代码保持不变就可以了，最后在 AndroidManifest.xml 中修改 ThirdActivity 的注册信息：

```
<activity android:name=".ThirdActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

我们在 ThirdActivity 的`<intent-filter>`中配置了当前活动能够响应的 `action` 是 `Intent.ACTION_VIEW` 的常量值，而 `category` 则毫无疑问指定了默认的 `category` 值，另外在`<data>`标签中我们通过 `android:scheme` 指定了数据的协议必须是 http 协议，这样 ThirdActivity 应该就和浏览器一样，能够响应一个打开网页的 Intent 了。让我们运行一下程序试试吧，在 FirstActivity 的界面点击一下按钮，结果如图 2.18 所示。

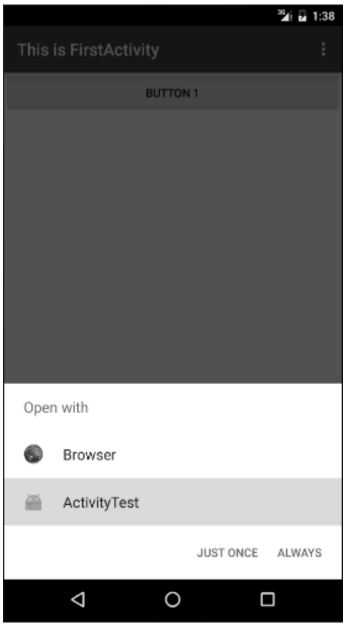


图 2.18　选择响应 Intent 的程序

可以看到，系统自动弹出了一个列表，显示了目前能够响应这个 Intent 的所有程序。选择 Browser 还会像之前一样打开浏览器，并显示百度的主页，而如果选择了 ActivityTest，则会启动 ThirdActivity。JUST ONCE 表示只是这次使用选择的程序打开，ALWAYS 则表示以后一直都使用这次选择的程序打开。需要注意的是，虽然我们声明了 ThirdActivity 是可以响应打开网页的 Intent 的，但实际上这个活动并没有加载并显示网页的功能，所以在真正的项目中尽量不要出现这种有可能误导用户的行为，不然会让用户对我们的应用产生负面的印象。

除了 http 协议外，我们还可以指定很多其他协议，比如 geo 表示显示地理位置、tel 表示拨打电话。下面的代码展示了如何在我们的程序中调用系统拨号界面。

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(Intent.ACTION_DIAL);
        intent.setData(Uri.parse("tel:10086"));
        startActivity(intent);
    }
});
```

首先指定了 Intent 的 `action` 是 `Intent.ACTION_DIAL`，这又是一个 Android 系统的内置动作。然后在 data 部分指定了协议是 tel，号码是 10086。重新运行一下程序，在 FirstActivity 的界面点击一下按钮，结果如图 2.19 所示。

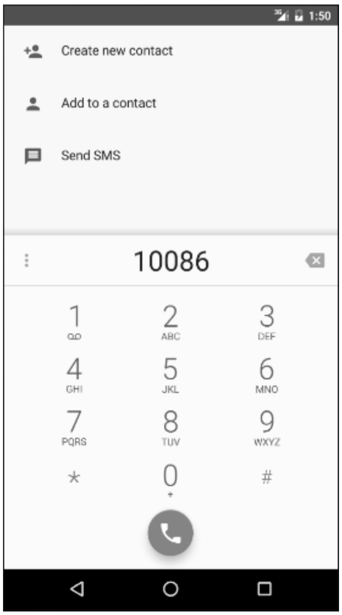


图 2.19 系统拨号界面

2.3.4　向下一个活动传递数据

经过前面几节的学习，你已经对 Intent 有了一定的了解。不过到目前为止，我们都只是简单地使用 Intent 来启动一个活动，其实 Intent 还可以在启动活动的时候传递数据，下面我们来一起看一下。

在启动活动时传递数据的思路很简单，Intent 中提供了一系列 `putExtra()`方法的重载，可以把我们想要传递的数据暂存在 Intent 中，启动了另一个活动后，只需要把这些数据再从 Intent 中取出就可以了。比如说 FirstActivity 中有一个字符串，现在想把这个字符串传递到 SecondActivity 中，你就可以这样编写：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String data = "Hello SecondActivity";
        Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
        intent.putExtra("extra_data", data);
        startActivity(intent);
    }
});
```

这里我们还是使用显式 Intent 的方式来启动 SecondActivity，并通过 `putExtra()`方法传递了一个字符串。注意这里 `putExtra()`方法接收两个参数，第一个参数是键，用于后面从 Intent 中取值，第二个参数才是真正要传递的数据。

然后我们在 SecondActivity 中将传递的数据取出，并打印出来，代码如下所示：

```
public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second_layout);
        Intent intent = getIntent();
        String data = intent.getStringExtra("extra_data");
        Log.d("SecondActivity", data);
    }

}
```

首先可以通过 `getIntent()`方法获取到用于启动 SecondActivity 的 Intent，然后调用 `getStringExtra()`方法，传入相应的键值，就可以得到传递的数据了。这里由于我们传递的是字符串，所以使用 `getStringExtra()`方法来获取传递的数据。如果传递的是整型数据，则使用 `getIntExtra()`方法；如果传递的是布尔型数据，则使用 `getBooleanExtra()`方法，以此类推。

重新运行程序，在 FirstActivity 的界面点击一下按钮会跳转到 SecondActivity，查看 logcat

打印信息，如图 2.20 所示。



图 2.20 SecondActivity 中的打印信息

可以看到，我们在 SecondActivity 中成功得到了从 FirstActivity 传递过来的数据。

2.3.5 返回数据给上一个活动

既然可以传递数据给下一个活动，那么能不能够返回数据给上一个活动呢？答案是肯定的。不过不同的是，返回上一个活动只需要按一下 Back 键就可以了，并没有一个用于启动活动的 Intent 来传递数据。通过查阅文档你会发现，Activity 中还有一个 `startActivityForResult()` 方法也是用于启动活动的，但这个方法期望在活动销毁的时候能够返回一个结果给上一个活动。毫无疑问，这就是我们所需要的。

`startActivityForResult()`方法接收两个参数，第一个参数还是 Intent，第二个参数是请求码，用于在之后的回调中判断数据的来源。我们还是来实战一下，修改 FirstActivity 中按钮的点击事件，代码如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
        startActivityForResult(intent, 1);
    }
});
```

这里我们使用了 `startActivityForResult()`方法来启动 SecondActivity，请求码只要是一个唯一值就可以了，这里传入了 1。接下来我们在 SecondActivity 中给按钮注册点击事件，并在点击事件中添加返回数据的逻辑，代码如下所示：

```
public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second_layout);
        Button button2 = (Button) findViewById(R.id.button_2);
        button2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent();
                intent.putExtra("data_return", "Hello FirstActivity");
                setResult(RESULT_OK, intent);
                finish();
```

```
            }
        });
    }

}
```

可以看到，我们还是构建了一个 Intent，只不过这个 Intent 仅仅是用于传递数据而已，它没有指定任何的“意图”。紧接着把要传递的数据存放在 Intent 中，然后调用了 `setResult()`方法。这个方法非常重要，是专门用于向上一个活动返回数据的。`setResult()`方法接收两个参数，第一个参数用于向上一个活动返回处理结果，一般只使用 `RESULT_OK` 或 `RESULT_CANCELED` 这两个值，第二个参数则把带有数据的 Intent 传递回去，然后调用了 `finish()`方法来销毁当前活动。

由于我们是使用 `startActivityForResult()`方法来启动 SecondActivity 的，在 SecondActivity 被销毁之后会回调上一个活动的 `onActivityResult()`方法，因此我们需要在 FirstActivity 中重写这个方法来得到返回的数据，如下所示：

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case 1:
            if (resultCode == RESULT_OK) {
                String returnedData = data.getStringExtra("data_return");
                Log.d("FirstActivity", returnedData);
            }
            break;
        default:
    }
}
```

`onActivityResult()`方法带有三个参数，第一个参数 `requestCode`，即我们在启动活动时传入的请求码。第二个参数 `resultCode`，即我们在返回数据时传入的处理结果。第三个参数 `data`，即携带着返回数据的 Intent。由于在一个活动中有可能调用 `startActivityForResult()`方法去启动很多不同的活动，每一个活动返回的数据都会回调到 `onActivityResult()`这个方法中，因此我们首先要做的就是通过检查 `requestCode` 的值来判断数据来源。确定数据是从 SecondActivity 返回的之后，我们再通过 `resultCode` 的值来判断处理结果是否成功。最后从 `data` 中取值并打印出来，这样就完成了向上一个活动返回数据的工作。

重新运行程序，在 FirstActivity 的界面点击按钮会打开 SecondActivity，然后在 SecondActivity 界面点击 Button 2 按钮会回到 FirstActivity，这时查看 logcat 的打印信息，如图 2.21 所示。



图 2.21 FirstActivity 中的打印信息

可以看到，SecondActivity 已经成功返回数据给 FirstActivity 了。

这时候你可能会问，如果用户在 SecondActivity 中并不是通过点击按钮，而是通过按下 Back 键回到 FirstActivity，这样数据不就没法返回了吗？没错，不过这种情况还是很好处理的，我们可以通过在 SecondActivity 中重写 `onBackPressed()`方法来解决这个问题，代码如下所示：

```
@Override
public void onBackPressed() {
    Intent intent = new Intent();
    intent.putExtra("data_return", "Hello FirstActivity");
    setResult(RESULT_OK, intent);
    finish();
}
```

这样的话，当用户按下 Back 键，就会去执行 `onBackPressed()`方法中的代码，我们在这里添加返回数据的逻辑就行了。

2.4 活动的生命周期

掌握活动的生命周期对任何 Android 开发者来说都非常重要，当你深入理解活动的生命周期之后，就可以写出更加连贯流畅的程序，并在如何合理管理应用资源方面发挥得游刃有余。你的应用程序将会拥有更好的用户体验。

2.4.1 返回栈

经过前面几节的学习，我相信你已经发现了这一点，Android 中的活动是可以层叠的。我们每启动一个新的活动，就会覆盖在原活动之上，然后点击 Back 键会销毁最上面的活动，下面的一个活动就会重新显示出来。

其实 Android 是使用任务（Task）来管理活动的，一个任务就是一组存放在栈里的活动的集合，这个栈也被称作返回栈（Back Stack）。栈是一种后进先出的数据结构，在默认情况下，每当我们启动了一个新的活动，它会在返回栈中入栈，并处于栈顶的位置。而每当我们按下 Back 键或调用 `finish()`方法去销毁一个活动时，处于栈顶的活动会出栈，这时前一个入栈的活动就会重新处于栈顶的位置。系统总是会显示处于栈顶的活动给用户。

示意图 2.22 展示了返回栈是如何管理活动入栈出栈操作的。

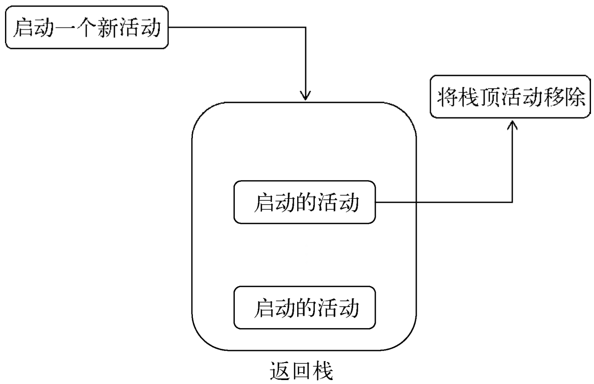


图 2.22　返回栈工作示意图

2.4.2　活动状态

每个活动在其生命周期中最多可能会有 4 种状态。

1. 运行状态

当一个活动位于返回栈的栈顶时，这时活动就处于运行状态。系统最不愿意回收的就是处于运行状态的活动，因为这会带来非常差的用户体验。

2. 暂停状态

当一个活动不再处于栈顶位置，但仍然可见时，这时活动就进入了暂停状态。你可能会觉得既然活动已经不在栈顶了，还怎么会可见呢？这是因为并不是每一个活动都会占满整个屏幕的，比如对话框形式的活动只会占用屏幕中间的部分区域，你很快就会在后面看到这种活动。处于暂停状态的活动仍然是完全存活着的，系统也不愿意去回收这种活动（因为它还是可见的，回收可见的东西都会在用户体验方面有不好的影响），只有在内存极低的情况下，系统才会去考虑回收这种活动。

3. 停止状态

当一个活动不再处于栈顶位置，并且完全不可见的时候，就进入了停止状态。系统仍然会为这种活动保存相应的状态和成员变量，但是这并不是完全可靠的，当其他地方需要内存时，处于停止状态的活动有可能会被系统回收。

4. 销毁状态

当一个活动从返回栈中移除后就变成了销毁状态。系统会最倾向于回收处于这种状态的活动，从而保证手机的内存充足。

2.4.3　活动的生存期

Activity 类中定义了 7 个回调方法，覆盖了活动生命周期的每一个环节，下面就来一一介绍这 7 个方法。

- ❑ onCreate()。这个方法你已经看到过很多次了，每个活动中我们都重写了这个方法，它会在活动第一次被创建的时候调用。你应该在这个方法中完成活动的初始化操作，比如说加载布局、绑定事件等。
- ❑ onStart()。这个方法在活动由不可见变为可见的时候调用。
- ❑ onResume()。这个方法在活动准备好和用户进行交互的时候调用。此时的活动一定位于返回栈的栈顶，并且处于运行状态。
- ❑ onPause()。这个方法在系统准备去启动或者恢复另一个活动的时候调用。我们通常会在这个方法中将一些消耗 CPU 的资源释放掉，以及保存一些关键数据，但这个方法的执行速度一定要快，不然会影响到新的栈顶活动的使用。
- ❑ onStop()。这个方法在活动完全不可见的时候调用。它和 onPause()方法的主要区别在于，如果启动的新活动是一个对话框式的活动，那么 onPause()方法会得到执行，而 onStop()方法并不会执行。
- ❑ onDestroy()。这个方法在活动被销毁之前调用，之后活动的状态将变为销毁状态。
- ❑ onRestart()。这个方法在活动由停止状态变为运行状态之前调用，也就是活动被重新启动了。

以上 7 个方法中除了 onRestart()方法，其他都是两两相对的，从而又可以将活动分为 3 种生存期。

- ❑ **完整生存期**。活动在 onCreate()方法和 onDestroy()方法之间所经历的，就是完整生存期。一般情况下，一个活动会在 onCreate()方法中完成各种初始化操作，而在 onDestroy()方法中完成释放内存的操作。
- ❑ **可见生存期**。活动在 onStart()方法和 onStop()方法之间所经历的，就是可见生存期。在可见生存期内，活动对于用户总是可见的，即便有可能无法和用户进行交互。我们可以通过这两个方法，合理地管理那些对用户可见的资源。比如在 onStart()方法中对资源进行加载，而在 onStop()方法中对资源进行释放，从而保证处于停止状态的活动不会占用过多内存。
- ❑ **前台生存期**。活动在 onResume()方法和 onPause()方法之间所经历的就是前台生存期。在前台生存期内，活动总是处于运行状态的，此时的活动是可以和用户进行交互的，我们平时看到和接触最多的也就是这个状态下的活动。

为了帮助你能够更好地理解，Android 官方提供了一张活动生命周期的示意图，如图 2.23 所示。

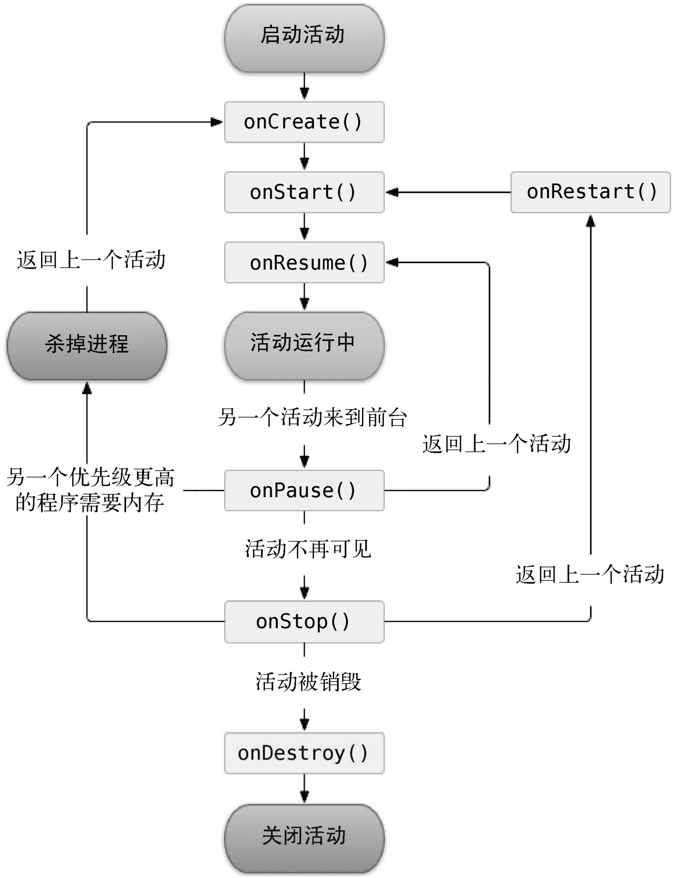


图 2.23　活动的生命周期

2.4.4　体验活动的生命周期

讲了这么多理论知识，也是时候该实战一下了，下面我们将通过一个实例，让你可以更加直观地体验活动的生命周期。

这次我们不准备在 ActivityTest 这个项目的基础上修改了，而是新建一个项目。因此，首先关闭 ActivityTest 项目，点击导航栏 File→Close Project。然后再新建一个 ActivityLifeCycleTest 项目，新建项目的过程你应该已经非常清楚了，不需要我再进行赘述，这次我们允许 Android Studio 帮我们自动创建活动和布局，这样可以省去不少工作，创建的活动名和布局名都使用默认值。

这样主活动就创建完成了，我们还需要分别再创建两个子活动——NormalActivity 和 DialogActivity，下面一步步来实现。

右击 com.example.activitylifecycletest 包→New→Activity→Empty Activity，新建 Normal-Activity，布局起名为 normal_layout。然后使用同样的方式创建 DialogActivity，布局起名为 dialog_layout。

现在编辑 normal_layout.xml 文件，将里面的代码替换成如下内容：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is a normal activity"
        />

</LinearLayout>
```

这个布局中我们就非常简单地使用了一个 TextView，用于显示一行文字，在下一章中你将会学到更多关于 TextView 的用法。

然后再编辑 dialog_layout.xml 文件，将里面的代码替换成如下内容：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is a dialog activity"
        />

</LinearLayout>
```

两个布局文件的代码几乎没有区别，只是显示的文字不同而已。

NormalActivity 和 DialogActivity 中的代码我们保持默认就好，不需要改动。

其实从名字上你就可以看出，这两个活动一个是普通的活动，一个是对话框式的活动。可是我们并没有修改活动的任何代码，两个活动的代码应该几乎是一模一样的，在哪里有体现出将活动设成对话框式的呢？别着急，下面我们马上开始设置。修改 AndroidManifest.xml 的`<activity>`标签的配置，如下所示：

```
<activity android:name=".NormalActivity">
</activity>
<activity android:name=".DialogActivity"
    android:theme="@style/Theme.AppCompat.Dialog">
</activity>
```

这里是两个活动的注册代码，但是 DialogActivity 的代码有些不同，我们给它使用了一个 `android:theme` 属性，这是用于给当前活动指定主题的，Android 系统内置有很多主题可以选择，当然我们也可以定制自己的主题，而这里`@style/Theme.AppCompat.Dialog` 则毫无疑问是让 DialogActivity 使用对话框式的主题。

接下来我们修改 activity_main.xml，重新定制主活动的布局，将里面的代码替换成如下内容：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/start_normal_activity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start NormalActivity" />

    <Button
        android:id="@+id/start_dialog_activity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start DialogActivity" />

</LinearLayout>
```

可以看到，我们在 LinearLayout 中加入了两个按钮，一个用于启动 NormalActivity，一个用于启动 DialogActivity。

最后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    public static final String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG,"onCreate");
        setContentView(R.layout.activity_main);
        Button startNormalActivity = (Button) findViewById(R.id.start_normal_
            activity);
        Button startDialogActivity = (Button) findViewById(R.id.start_dialog_
            activity);
        startNormalActivity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, NormalActivity.class);
                startActivity(intent);
            }
        });
        startDialogActivity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                Intent intent = new Intent(MainActivity.this, DialogActivity.class);
                startActivity(intent);
            }
        });
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.d(TAG, "onStart");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.d(TAG, "onResume");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.d(TAG, "onPause");
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.d(TAG, "onStop");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d(TAG, "onRestart");
    }

}
```

在 `onCreate()`方法中，我们分别为两个按钮注册了点击事件，点击第一个按钮会启动 NormalActivity，点击第二个按钮会启动 DialogActivity。然后在 Activity 的 7 个回调方法中分别打印了一句话，这样就可以通过观察日志的方式来更直观地理解活动的生命周期。

现在运行程序，效果如图 2.24 所示。



图 2.24　MainActivity 界面

这时观察 logcat 中的打印日志，如图 2.25 所示。



图 2.25　启动程序时的打印日志

可以看到，当 MainActivity 第一次被创建时会依次执行 `onCreate()`、`onStart()`和`onResume()`方法。然后点击第一个按钮，启动 NormalActivity，如图 2.26 所示。

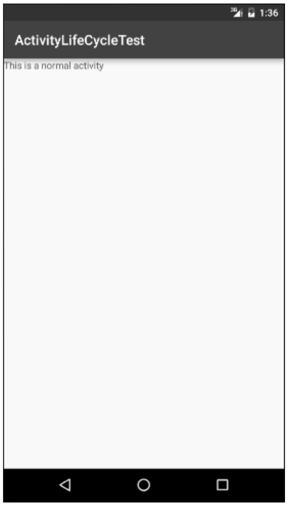


图 2.26　NormalActivity 界面

此时的打印信息如图 2.27 所示。

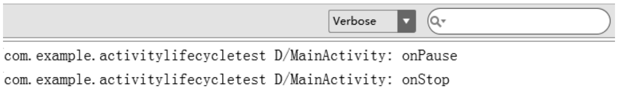


图 2.27　打开 NormalActivity 时的打印日志

由于 NormalActivity 已经把 MainActivity 完全遮挡住，因此 `onPause()`和 `onStop()`方法都会得到执行。然后按下 Back 键返回 MainActivity，打印信息如图 2.28 所示。



图 2.28　返回 MainActivity 的打印日志

由于之前 MainActivity 已经进入了停止状态，所以 `onRestart()`方法会得到执行，之后又会依次执行 `onStart()`和 `onResume()`方法。注意此时 `onCreate()`方法不会执行，因为 MainActivity 并没有重新创建。

然后再点击第二个按钮，启动 DialogActivity，如图 2.29 所示。

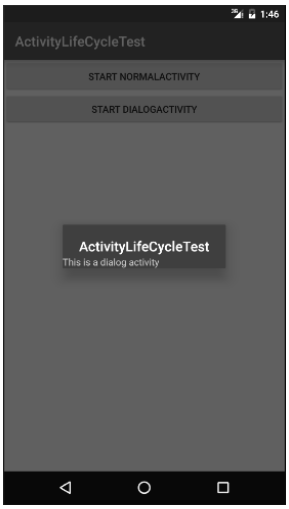


图 2.29　DialogActivity 界面

此时观察打印信息，如图 2.30 所示。



图 2.30　打开 DialogActivity 时的打印日志

可以看到，只有 onPause()方法得到了执行，onStop()方法并没有执行，这是因为 DialogActivity 并没有完全遮挡住 MainActivity，此时 MainActivity 只是进入了暂停状态，并没有进入停止状态。相应地，按下 Back 键返回 MainActivity 也应该只有 onResume()方法会得到执行，如图 2.31 所示。



图 2.31　再次返回 MainActivity 的打印日志

最后在 MainActivity 按下 Back 键退出程序，打印信息如图 2.32 所示。



图 2.32　退出程序时的打印日志

依次会执行 onPause()、onStop()和 onDestroy()方法，最终销毁 MainActivity。

这样活动完整的生命周期你已经体验了一遍，是不是理解得更加深刻了？

2.4.5　活动被回收了怎么办

前面我们已经说过，当一个活动进入到了停止状态，是有可能被系统回收的。那么想象以下场景：应用中有一个活动 A，用户在活动 A 的基础上启动了活动 B，活动 A 就进入了停止状态，这个时候由于系统内存不足，将活动 A 回收掉了，然后用户按下 Back 键返回活动 A，会出现什么情况呢？其实还是会正常显示活动 A 的，只不过这时并不会执行 onRestart()方法，而是会执行活动 A 的 onCreate()方法，因为活动 A 在这种情况下会被重新创建一次。

这样看上去好像一切正常，可是别忽略了一个重要问题，活动 A 中是可能存在临时数据和状态的。打个比方，MainActivity 中有一个文本输入框，现在你输入了一段文字，然后启动 NormalActivity，这时 MainActivity 由于系统内存不足被回收掉，过了一会你又点击了 Back 键回到 MainActivity，你会发现刚刚输入的文字全部都没了，因为 MainActivity 被重新创建了。

如果我们的应用出现了这种情况，是会严重影响用户体验的，所以必须要想想办法解决这个问题。查阅文档可以看出，Activity 中还提供了一个 onSaveInstanceState()回调方法，这个方法可以保证在活动被回收之前一定会被调用，因此我们可以通过这个方法来解决活动被回收时临时数据得不到保存的问题。

onSaveInstanceState()方法会携带一个 Bundle 类型的参数，Bundle 提供了一系列的方法用于保存数据，比如可以使用 putString()方法保存字符串，使用 putInt()方法保存整型数据，以此类推。每个保存方法需要传入两个参数，第一个参数是键，用于后面从 Bundle 中取值，

第二个参数是真正要保存的内容。

在 MainActivity 中添加如下代码就可以将临时数据进行保存：

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    String tempData = "Something you just typed";
    outState.putString("data_key", tempData);
}
```

数据是已经保存下来了，那么我们应该在哪里进行恢复呢？细心的你也许早就发现，我们一直使用的 `onCreate()`方法其实也有一个 `Bundle` 类型的参数。这个参数在一般情况下都是 `null`，但是如果在活动被系统回收之前有通过 `onSaveInstanceState()`方法来保存数据的话，这个参数就会带有之前所保存的全部数据，我们只需要再通过相应的取值方法将数据取出即可。

修改 MainActivity 的 `onCreate()`方法，如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate");
    setContentView(R.layout.activity_main);
    if (savedInstanceState != null) {
        String tempData = savedInstanceState.getString("data_key");
        Log.d(TAG, tempData);
    }
    ...
}
```

取出值之后再做相应的恢复操作就可以了，比如说将文本内容重新赋值到文本输入框上，这里我们只是简单地打印一下。

不知道你有没有察觉，使用 `Bundle` 来保存和取出数据是不是有些似曾相识呢？没错！我们在使用 Intent 传递数据时也是用的类似的方法。这里跟你提醒一点，Intent 还可以结合 `Bundle` 一起用于传递数据，首先可以把需要传递的数据都保存在 `Bundle` 对象中，然后再将 `Bundle` 对象存放在 Intent 里。到了目标活动之后先从 Intent 中取出 `Bundle`，再从 `Bundle` 中一一取出数据。具体的代码我就不写了，要学会举一反三哦。

2.5　活动的启动模式

活动的启动模式对你来说应该是个全新的概念，在实际项目中我们应该根据特定的需求为每个活动指定恰当的启动模式。启动模式一共有 4 种，分别是 standard、singleTop、singleTask 和 singleInstance，可以在 AndroidManifest.xml 中通过给`<activity>`标签指定 `android:launchMode` 属性来选择启动模式。下面我们来逐个进行学习。

2.5.1　standard

standard 是活动默认的启动模式，在不进行显式指定的情况下，所有活动都会自动使用这种启动模式。因此，到目前为止我们写过的所有活动都是使用的 standard 模式。经过上一节的学习，你已经知道了 Android 是使用返回栈来管理活动的，在 standard 模式（即默认情况）下，每当启动一个新的活动，它就会在返回栈中入栈，并处于栈顶的位置。对于使用 standard 模式的活动，系统不会在乎这个活动是否已经在返回栈中存在，每次启动都会创建该活动的一个新的实例。

我们现在通过实践来体会一下 standard 模式，这次还是准备在 ActivityTest 项目的基础上修改，首先关闭 ActivityLifeCycleTest 项目，打开 ActivityTest 项目。

修改 FirstActivity 中 onCreate()方法的代码，如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("FirstActivity", this.toString());
    setContentView(R.layout.first_layout);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(FirstActivity.this, FirstActivity.class);
            startActivity(intent);
        }
    });
}
```

代码看起来有些奇怪吧，在 FirstActivity 的基础上启动 FirstActivity。从逻辑上来讲这确实没什么意义，不过我们的重点在于研究 standard 模式，因此不必在意这段代码有什么实际用途。另外我们还在 onCreate()方法中添加了一行打印信息，用于打印当前活动的实例。

现在重新运行程序，然后在 FirstActivity 界面连续点击两次按钮，可以看到 logcat 中打印信息如图 2.33 所示。



图 2.33　standard 模式下的打印日志

从打印信息中我们就可以看出，每点击一次按钮就会创建出一个新的 FirstActivity 实例。此时返回栈中也会存在 3 个 FirstActivity 的实例，因此你需要连按 3 次 Back 键才能退出程序。

standard 模式的原理示意图，如图 2.34 所示。

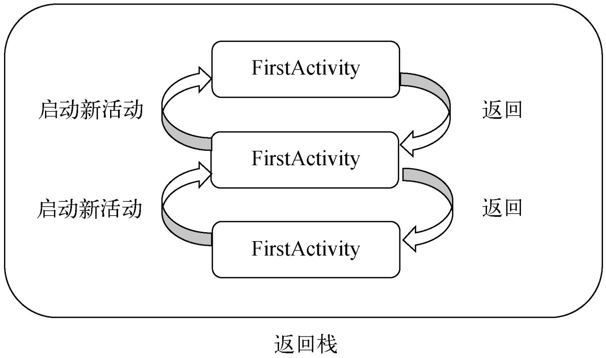


图 2.34　standard 模式示意图

2.5.2　singleTop

可能在有些情况下，你会觉得 standard 模式不太合理。活动明明已经在栈顶了，为什么再次启动的时候还要创建一个新的活动实例呢？别着急，这只是系统默认的一种启动模式而已，你完全可以根据自己的需要进行修改，比如说使用 singleTop 模式。当活动的启动模式指定为 singleTop，在启动活动时如果发现返回栈的栈顶已经是该活动，则认为可以直接使用它，不会再创建新的活动实例。

我们还是通过实践来体会一下，修改 AndroidManifest.xml 中 FirstActivity 的启动模式，如下所示：

```
<activity
    android:name=".FirstActivity"
    android:launchMode="singleTop"
    android:label="This is FirstActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

然后重新运行程序，查看 logcat 会看到已经创建了一个 FirstActivity 的实例，如图 2.35 所示。

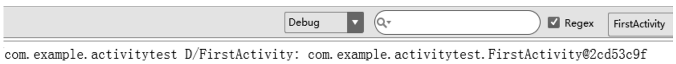


图 2.35　singleTop 模式下的打印日志

但是之后不管你点击多少次按钮都不会再有新的打印信息出现，因为目前 FirstActivity 已经处于返回栈的栈顶，每当想要再启动一个 FirstActivity 时都会直接使用栈顶的活动，因此 FirstActivity 也只会有一个实例，仅按一次 Back 键就可以退出程序。

不过当 FirstActivity 并未处于栈顶位置时，这时再启动 FirstActivity，还是会创建新的实例的。

下面我们来实验一下，修改 FirstActivity 中 `onCreate()`方法的代码，如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("FirstActivity", this.toString());
    setContentView(R.layout.first_layout);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
            startActivity(intent);
        }
    });
}
```

这次我们点击按钮后启动的是 SecondActivity。然后修改 SecondActivity 中 `onCreate()`方法的代码，如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("SecondActivity", this.toString());
    setContentView(R.layout.second_layout);
    Button button2 = (Button) findViewById(R.id.button_2);
    button2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(SecondActivity.this, FirstActivity.class);
            startActivity(intent);
        }
    });
}
```

我们在 SecondActivity 中的按钮点击事件里又加入了启动 FirstActivity 的代码。现在重新运行程序，在 FirstActivity 界面点击按钮进入到 SecondActivity，然后在 SecondActivity 界面点击按钮，又会重新进入到 FirstActivity。

查看 logcat 中的打印信息，如图 2.36 所示。



图 2.36　singleTop 模式下的打印日志

可以看到系统创建了两个不同的 FirstActivity 实例，这是由于在 SecondActivity 中再次启动 FirstActivity 时，栈顶活动已经变成了 SecondActivity，因此会创建一个新的 FirstActivity 实例。现在按下 Back 键会返回到 SecondActivity，再次按下 Back 键又会回到 FirstActivity，再按一次

Back 键才会退出程序。

singleTop 模式的原理示意图，如图 2.37 所示。

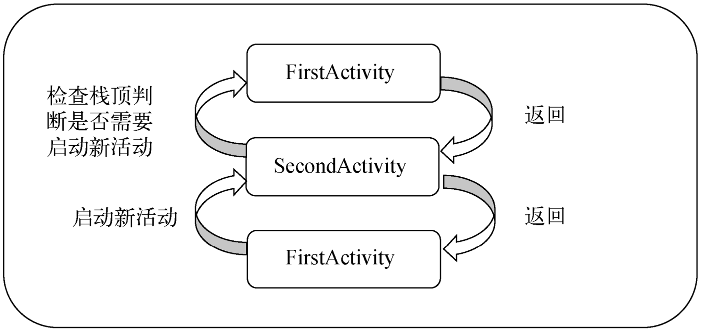


图 2.37　singleTop 模式示意图

2.5.3　singleTask

使用 singleTop 模式可以很好地解决重复创建栈顶活动的问题，但是正如你在上一节所看到的，如果该活动并没有处于栈顶的位置，还是可能会创建多个活动实例的。那么有没有什么办法可以让某个活动在整个应用程序的上下文中只存在一个实例呢？这就要借助 singleTask 模式来实现了。当活动的启动模式指定为 singleTask，每次启动该活动时系统首先会在返回栈中检查是否存在该活动的实例，如果发现已经存在则直接使用该实例，并把在这个活动之上的所有活动统统出栈，如果没有发现就会创建一个新的活动实例。

我们还是通过代码来更加直观地理解一下。修改 AndroidManifest.xml 中 FirstActivity 的启动模式：

```
<activity
    android:name=".FirstActivity"
    android:launchMode="singleTask"
    android:label="This is FirstActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

然后在 FirstActivity 中添加 `onRestart()`方法，并打印日志：

```
@Override
protected void onRestart() {
    super.onRestart();
    Log.d("FirstActivity", "onRestart");
}
```

最后在 SecondActivity 中添加 `onDestroy()`方法，并打印日志：

```
@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d("SecondActivity", "onDestroy");
}
```

现在重新运行程序，在 FirstActivity 界面点击按钮进入到 SecondActivity，然后在 SecondActivity 界面点击按钮，又会重新进入到 FirstActivity。

查看 logcat 中的打印信息，如图 2.38 所示。

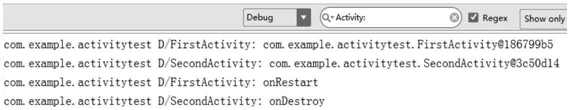


图 2.38　singleTask 模式下的打印日志

其实从打印信息中就可以明显看出了，在 SecondActivity 中启动 FirstActivity 时，会发现返回栈中已经存在一个 FirstActivity 的实例，并且是在 SecondActivity 的下面，于是 SecondActivity 会从返回栈中出栈，而 FirstActivity 重新成为了栈顶活动，因此 FirstActivity 的 `onRestart()`方法和 SecondActivity 的 `onDestroy()`方法会得到执行。现在返回栈中应该只剩下一个 FirstActivity 的实例了，按一下 Back 键就可以退出程序。

singleTask 模式的原理示意图，如图 2.39 所示。

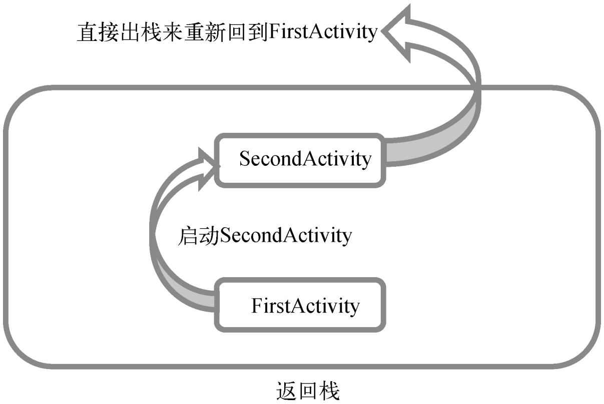


图 2.39　singleTask 模式示意图

2.5.4　singleInstance

singleInstance 模式应该算是 4 种启动模式中最特殊也最复杂的一个了，你也需要多花点功夫来理解这个模式。不同于以上 3 种启动模式，指定为 singleInstance 模式的活动会启用一个新的返

回栈来管理这个活动（其实如果 singleTask 模式指定了不同的 taskAffinity，也会启动一个新的返回栈）。那么这样做有什么意义呢？想象以下场景，假设我们的程序中有一个活动是允许其他程序调用的，如果我们想实现其他程序和我们的程序可以共享这个活动的实例，应该如何实现呢？使用前面 3 种启动模式肯定是做不到的，因为每个应用程序都会有自己的返回栈，同一个活动在不同的返回栈中入栈时必然是创建了新的实例。而使用 singleInstance 模式就可以解决这个问题，在这种模式下会有一个单独的返回栈来管理这个活动，不管是哪个应用程序来访问这个活动，都共用的同一个返回栈，也就解决了共享活动实例的问题。

为了帮助你更好地理解这种启动模式，我们还是来实践一下。修改 AndroidManifest.xml 中 SecondActivity 的启动模式：

```
<activity android:name=".SecondActivity"
    android:launchMode="singleInstance">
    <intent-filter>
        <action android:name="com.example.activitytest.ACTION_START" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="com.example.activitytest.MY_CATEGORY" />
    </intent-filter>
</activity>
```

我们先将 SecondActivity 的启动模式指定为 singleInstance，然后修改 FirstActivity 中 `onCreate()`方法的代码：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("FirstActivity", "Task id is " + getTaskId());
    setContentView(R.layout.first_layout);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
            startActivity(intent);
        }
    });
}
```

在 `onCreate()`方法中打印了当前返回栈的 `id`。然后修改 SecondActivity 中 `onCreate()`方法的代码：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("SecondActivity", "Task id is " + getTaskId());
    setContentView(R.layout.second_layout);
    Button button2 = (Button) findViewById(R.id.button_2);
    button2.setOnClickListener(new View.OnClickListener() {
        @Override
```

```
            public void onClick(View v) {
                Intent intent = new Intent(SecondActivity.this, ThirdActivity.class);
                startActivity(intent);
            }
        });
    }
```

同样在 `onCreate()`方法中打印了当前返回栈的 `id`，然后又修改了按钮点击事件的代码，用于启动 ThirdActivity。最后修改 ThirdActivity 中 `onCreate()`方法的代码：

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("ThirdActivity", "Task id is " + getTaskId());
        setContentView(R.layout.third_layout);
    }
```

仍然是在 `onCreate()`方法中打印了当前返回栈的 `id`。现在重新运行程序，在 FirstActivity 界面点击按钮进入到 SecondActivity，然后在 SecondActivity 界面点击按钮进入到 ThirdActivity。

查看 logcat 中的打印信息，如图 2.40 所示。

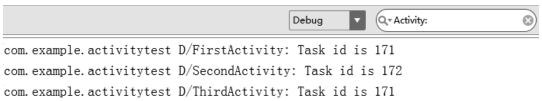


图 2.40　singleInstance 模式下的打印日志

可以看到，SecondActivity 的 `Task id` 不同于 FirstActivity 和 ThirdActivity，这说明 SecondActivity 确实是存放在一个单独的返回栈里的，而且这个栈中只有 SecondActivity 这一个活动。

然后我们按下 Back 键进行返回，你会发现 ThirdActivity 竟然直接返回到了 FirstActivity，再按下 Back 键又会返回到 SecondActivity，再按下 Back 键才会退出程序，这是为什么呢？其实原理很简单，由于 FirstActivity 和 ThirdActivity 是存放在同一个返回栈里的，当在 ThirdActivity 的界面按下 Back 键，ThirdActivity 会从返回栈中出栈，那么 FirstActivity 就成为了栈顶活动显示在界面上，因此也就出现了从 ThirdActivity 直接返回到 FirstActivity 的情况。然后在 FirstActivity 界面再次按下 Back 键，这时当前的返回栈已经空了，于是就显示了另一个返回栈的栈顶活动，即 SecondActivity。最后再次按下 Back 键，这时所有返回栈都已经空了，也就自然退出了程序。

singleInstance 模式的原理示意图，如图 2.41 所示。

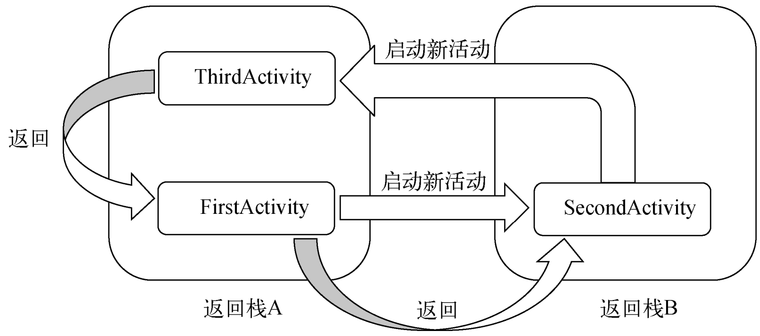


图 2.41 singleInstance 模式示意图

2.6 活动的最佳实践

你已经掌握了关于活动非常多的知识，不过恐怕离能够完全灵活运用还有一段距离。虽然知识点只有这么多，但运用的技巧却是多种多样的。所以，在这里我准备教你几种关于活动的最佳实践技巧，这些技巧在你以后的开发工作当中将会非常有用。

2.6.1 知晓当前是在哪一个活动

这个技巧将教会你如何根据程序当前的界面就能判断出这是哪一个活动。可能你会觉得挺纳闷的，我自己写的代码怎么会不知道这是哪一个活动呢？很不幸的是，在你真正进入到企业之后，更有可能的是接手一份别人写的代码，因为你刚进公司就正好有一个新项目启动的概率并不高。阅读别人的代码时有一个很头疼的问题，就是当你需要在某个界面上修改一些非常简单的东西时，却半天找不到这个界面对应的活动是哪一个。学会了本节的技巧之后，这对你来说就再也不是难题了。

我们还是在 ActivityTest 项目的基础上修改，首先需要新建一个 `BaseActivity` 类。右击 com.example.activitytest 包→New→Java Class，在弹出的窗口出输入 BaseActivity，如图 2.42 所示。

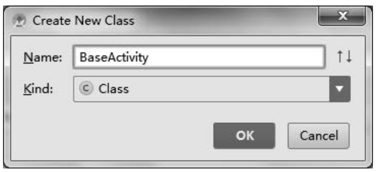


图 2.42 创建 `BaseActivity` 类

注意这里 `BaseActivity` 和普通活动的创建方式并不一样，因为我们不需要让 `BaseActivity` 在 AndroidManifest.xml 中注册，所以选择创建一个普通的 Java 类就可以了。然后让 `BaseActivity` 继承自 `AppCompatActivity`，并重写 `onCreate()`方法，如下所示：

```
public class BaseActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("BaseActivity", getClass().getSimpleName());
    }
}
```

我们在 onCreate()方法中获取了当前实例的类名，并通过 Log 打印了出来。

接下来我们需要让 BaseActivity 成为 ActivityTest 项目中所有活动的父类。修改 FirstActivity、SecondActivity 和 ThirdActivity 的继承结构，让它们不再继承自 AppCompatActivity，而是继承自 BaseActivity。而由于 BaseActivity 又是继承自 AppCompatActivity 的，所以项目中所有活动的现有功能并不受影响，它们仍然完全继承了 Activity 中的所有特性。

现在重新运行程序，然后通过点击按钮分别进入到 FirstActivity、SecondActivity 和 ThirdActivity 的界面，这时观察 logcat 中的打印信息，如图 2.43 所示。

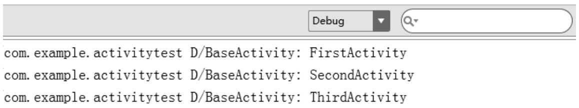


图 2.43　BaseActivity 中的打印日志

现在每当我们进入到一个活动的界面，该活动的类名就会被打印出来，这样我们就可以时时刻刻知晓当前界面对应的是哪一个活动了。

2.6.2　随时随地退出程序

如果目前你手机的界面还停留在 ThirdActivity，你会发现当前想退出程序是非常不方便的，需要连按 3 次 Back 键才行。按 Home 键只是把程序挂起，并没有退出程序。其实这个问题就足以引起你的思考，如果我们的程序需要一个注销或者退出的功能该怎么办呢？必须要有一个随时随地都能退出程序的方案才行。

其实解决思路也很简单，只需要用一个专门的集合类对所有的活动进行管理就可以了，下面我们就来实现一下。

新建一个 ActivityCollector 类作为活动管理器，代码如下所示：

```
public class ActivityCollector {

    public static List<Activity> activities = new ArrayList<>();

    public static void addActivity(Activity activity) {
        activities.add(activity);
    }
```

```
    public static void removeActivity(Activity activity) {
        activities.remove(activity);
    }

    public static void finishAll() {
        for (Activity activity : activities) {
            if (!activity.isFinishing()) {
                activity.finish();
            }
        }
        activities.clear();
    }

}
```

在活动管理器中，我们通过一个 List 来暂存活动，然后提供了一个 addActivity()方法用于向 List 中添加一个活动，提供了一个 removeActivity()方法用于从 List 中移除活动，最后提供了一个 finishAll()方法用于将 List 中存储的活动全部销毁掉。

接下来修改 BaseActivity 中的代码，如下所示：

```
public class BaseActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("BaseActivity", getClass().getSimpleName());
        ActivityCollector.addActivity(this);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        ActivityCollector.removeActivity(this);
    }

}
```

在 BaseActivity 的 onCreate()方法中调用了 ActivityCollector 的 addActivity()方法，表明将当前正在创建的活动添加到活动管理器里。然后在 BaseActivity 中重写 onDestroy()方法，并调用了 ActivityCollector 的 removeActivity()方法，表明将一个马上要销毁的活动从活动管理器里移除。

从此以后，不管你想在什么地方退出程序，只需要调用 ActivityCollector.finishAll()方法就可以了。例如在 ThirdActivity 界面想通过点击按钮直接退出程序，只需将代码改成如下所示：

```
public class ThirdActivity extends BaseActivity {

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("ThirdActivity", "Task id is " + getTaskId());
        setContentView(R.layout.third_layout);
        Button button3 = (Button) findViewById(R.id.button_3);
        button3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ActivityCollector.finishAll();
            }
        });

    }
}
```

当然你还可以在销毁所有活动的代码后面再加上杀掉当前进程的代码，以保证程序完全退出，杀掉进程的代码如下所示：

```
android.os.Process.killProcess(android.os.Process.myPid());
```

其中，`killProcess()`方法用于杀掉一个进程，它接收一个进程 `id` 参数，我们可以通过 `myPid()`方法来获得当前程序的进程 `id`。需要注意的是，`killProcess()`方法只能用于杀掉当前程序的进程，我们不能使用这个方法去杀掉其他程序。

2.6.3　启动活动的最佳写法

启动活动的方法相信你已经非常熟悉了，首先通过 Intent 构建出当前的“意图”，然后调用 `startActivity()`或 `startActivityForResult()`方法将活动启动起来，如果有数据需要从一个活动传递到另一个活动，也可以借助 Intent 来完成。

假设 SecondActivity 中需要用到两个非常重要的字符串参数，在启动 SecondActivity 的时候必须要传递过来，那么我们很容易会写出如下代码：

```
Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
intent.putExtra("param1", "data1");
intent.putExtra("param2", "data2");
startActivity(intent);
```

这样写是完全正确的，不管是从语法上还是规范上，只是在真正的项目开发中经常会有对接的问题出现。比如 SecondActivity 并不是由你开发的，但现在你负责的部分需要有启动 SecondActivity 这个功能，而你却不清楚启动这个活动需要传递哪些数据。这时无非就有两种办法，一个是你自己去阅读 SecondActivity 中的代码，二是询问负责编写 SecondActivity 的同事。你会不会觉得很麻烦呢？其实只需要换一种写法，就可以轻松解决掉上面的窘境。

修改 SecondActivity 中的代码，如下所示：

```
public class SecondActivity extends BaseActivity {
```

```
    public static void actionStart(Context context, String data1, String data2) {
        Intent intent = new Intent(context, SecondActivity.class);
        intent.putExtra("param1", data1);
        intent.putExtra("param2", data2);
        context.startActivity(intent);
    }
    ...
}
```

我们在 SecondActivity 中添加了一个 `actionStart()`方法，在这个方法中完成了 Intent 的构建，另外所有 SecondActivity 中需要的数据都是通过 `actionStart()`方法的参数传递过来的，然后把它们存储到 Intent 中，最后调用 `startActivity()`方法启动 SecondActivity。

这样写的好处在哪里呢？最重要的一点就是一目了然，SecondActivity 所需要的数据在方法参数中全部体现出来了，这样即使不用阅读 SecondActivity 中的代码，不去询问负责编写 SecondActivity 的同事，你也可以非常清晰地知道启动 SecondActivity 需要传递哪些数据。另外，这样写还简化了启动活动的代码，现在只需要一行代码就可以启动 SecondActivity，如下所示：

```
button1.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        SecondActivity.actionStart(FirstActivity.this, "data1", "data2");
    }
});
```

养成一个良好的习惯，给你编写的每个活动都添加类似的启动方法，这样不仅可以让启动活动变得非常简单，还可以节省不少你同事过来询问你的时间。

2.7　小结与点评

真是好疲惫啊！没错，学习了这么多的东西不疲惫才怪呢。但是，你内心那种掌握了知识的喜悦感相信也是无法掩盖的。本章的收获非常多啊，不管是理论型还是实践型的东西都涉及了，从活动的基本用法，到启动活动和传递数据的方式，再到活动的生命周期，以及活动的启动模式，你几乎已经学会了关于活动所有重要的知识点。另外在本章的最后，还学习了几种可以应用在活动中的最佳实践技巧，毫不夸张地说，你在 Android 活动方面已经算是一个小高手了。

不过你的 Android 旅途才刚刚开始呢，后面需要学习的东西还很多，也许会比现在还累，一定要做好心理准备哦。总体来说，我给你现在的状态打满分，毕竟你已经学会了那么多的东西，也是时候放松一下了。自己适当控制一下休息的时间，然后我们继续前进吧！

第 3 章

软件也要拼脸蛋——UI 开发的点点滴滴

我一直都认为程序员在软件的审美方面普遍都比较差，至少我个人就是如此。如果说要追究其根本原因，我觉得这是由程序员的工作性质所导致的。每当我们看到一个软件时，不会像普通用户那样仅仅是关注一下它的界面和功能，而是会不自觉地思考这些功能是如何实现的。很多在普通用户看来理所应当的功能，背后可能却需要非常复杂的逻辑来完成，以至于当别人唾骂一句“这软件做得真丑”的时候，我们还可能赞叹一句“这功能做得好牛啊”！

不过缺乏审美观毕竟不是一件值得炫耀的事情，在软件开发过程中，界面设计和功能开发同样重要。界面美观的应用程序不仅可以大大增加用户粘性，还能帮我们吸引到更多的新用户。而 Android 也给我们提供了大量的 UI 开发工具，只要合理地使用它们，就可以编写出各种各样漂亮的界面。

在这里，我无法教会你如何提升自己的审美观，但我可以教会你怎样使用 Android 提供的 UI 开发工具来编写程序界面。你在上一章中反反复复地使用那几个按钮，想必都快要吐了吧，本章我们就来学习更多的 UI 开发方面的知识。

3.1 如何编写程序界面

Android 中有多种编写程序界面的方式可供选择。Android Studio 和 Eclipse 中都提供了相应的可视化编辑器，允许使用拖放控件的方式来编写布局，并能在视图上直接修改控件的属性。不过我并不推荐你使用这种方式来编写界面，因为可视化编辑工具并不利于你去真正了解界面背后的实现原理。通过这种方式制作出的界面通常不具有很好的屏幕适配性，而且当需要编写较为复杂的界面时，可视化编辑工具将很难胜任。因此本书中所有的界面都将通过最基本的方式去实现，即编写 XML 代码。等你完全掌握了使用 XML 来编写界面的方法之后，不管是进行高复杂度的界面实现，还是分析和修改当前现有界面，对你来说都将是手到擒来。

讲了这么多理论的东西，也是时候学习一下到底如何编写程序界面了，下面我们就从 Android 中几种常见的控件开始吧。

3.2 常用控件的使用方法

Android 提供了大量的 UI 控件，合理地使用这些控件就可以非常轻松地编写出相当不错的界面，下面我们就挑选几种常用的控件，详细介绍一下它们的使用方法。

首先新建一个 UIWidgetTest 项目，简单起见，我们还是允许 Android Studio 自动创建活动，活动名和布局名都使用默认值。

3.2.1 TextView

TextView 可以说是 Android 中最简单的一个控件了，你在前面其实已经和它打过一些交道了。它主要用于在界面上显示一段文本信息，比如你在第 1 章看到的“Hello world！”。下面我们就来看一看关于 TextView 的更多用法。

修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is TextView" />

</LinearLayout>
```

外面的 LinearLayout 先忽略不看，在 TextView 中我们使用 `android:id` 给当前控件定义了一个唯一标识符，这个属性在上一章中已经讲解过了。然后使用 `android:layout_width` 和 `android:layout_height` 指定了控件的宽度和高度。Android 中所有的控件都具有这两个属性，可选值有 3 种：`match_parent`、`fill_parent` 和 `wrap_content`。其中 `match_parent` 和 `fill_parent` 的意义相同，现在官方更加推荐使用 `match_parent`。`match_parent` 表示让当前控件的大小和父布局的大小一样，也就是由父布局来决定当前控件的大小。`wrap_content` 表示让当前控件的大小能够刚好包含住里面的内容，也就是由控件内容决定当前控件的大小。所以上面的代码就表示让 TextView 的宽度和父布局一样宽，也就是手机屏幕的宽度，让 TextView 的高度足够包含住里面的内容就行。当然除了使用上述值，你也可以对控件的宽和高指定一个固定的大小，但是这样做有时会在不同手机屏幕的适配方面出现问题。

接下来我们通过 `android:text` 指定 TextView 中显示的文本内容，现在运行程序，效果如图 3.1 所示。

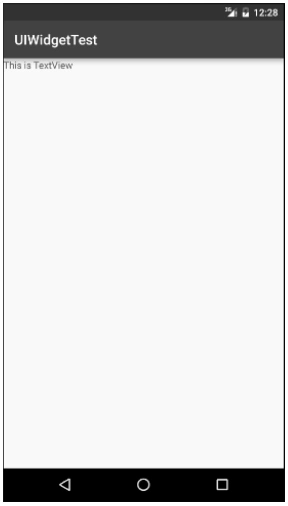


图 3.1　TextView 运行效果

虽然指定的文本内容正常显示了，不过我们好像没看出来 TextView 的宽度是和屏幕一样宽的。其实这是由于 TextView 中的文字默认是居左上角对齐的，虽然 TextView 的宽度充满了整个屏幕，可是由于文字内容不够长，所以从效果上完全看不出来。现在我们修改 TextView 的文字对齐方式，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="This is TextView" />

</LinearLayout>
```

我们使用 `android:gravity` 来指定文字的对齐方式，可选值有 `top`、`bottom`、`left`、`right`、`center` 等，可以用“|”来同时指定多个值，这里我们指定的 `center`，效果等同于 `center_vertical|center_horizontal`，表示文字在垂直和水平方向都居中对齐。现在重新运行程序，效果如图 3.2 所示。

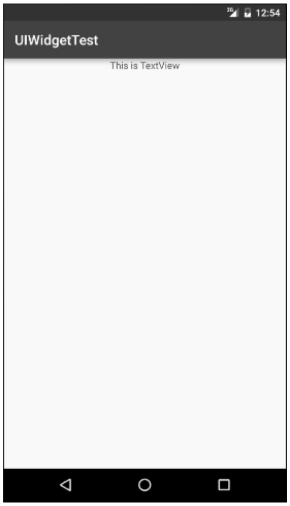


图 3.2　TextView 居中效果

这也说明了 TextView 的宽度确实是和屏幕宽度一样的。

另外我们还可以对 TextView 中文字的大小和颜色进行修改，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="24sp"
        android:textColor="#00ff00"
        android:text="This is TextView" />

</LinearLayout>
```

通过 `android:textSize` 属性可以指定文字的大小，通过 `android:textColor` 属性可以指定文字的颜色，在 Android 中字体大小使用 sp 作为单位。重新运行程序，效果如图 3.3 所示。

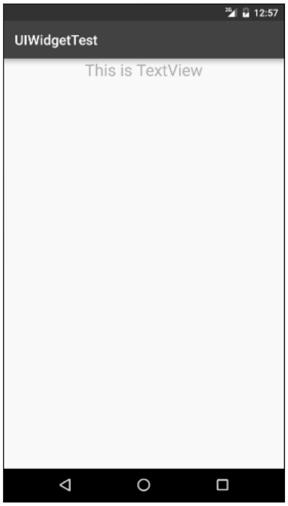


图 3.3　改变 TextView 文字大小和颜色的效果

当然 TextView 中还有很多其他的属性，这里就不再一一介绍了，用到的时候去查阅文档就可以了。

3.2.2　Button

Button 是程序用于和用户进行交互的一个重要控件，相信你对这个控件已经非常熟悉了，因为我们在上一章用了太多次 Button。它可配置的属性和 TextView 是差不多的，我们可以在 activity_main.xml 中这样加入 Button：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />

</LinearLayout>
```

加入 Button 之后的界面如图 3.4 所示。

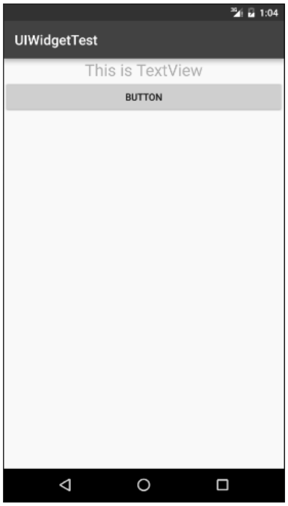


图 3.4 Button 运行效果

细心的你可能会留意到，我们在布局文件里面设置的文字是“Button”，但最终的显示结果却是“BUTTON”。这是由于系统会对 Button 中的所有英文字母自动进行大写转换，如果这不是你想要的效果，可以使用如下配置来禁用这一默认特性：

```
<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button"
    android:textAllCaps="false" />
```

接下来我们可以在 MainActivity 中为 Button 的点击事件注册一个监听器，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 在此处添加逻辑
            }
        });
    }
}
```

这样每当点击按钮时，就会执行监听器中的 `onClick()`方法，我们只需要在这个方法中加入待处理的逻辑就行了。如果你不喜欢使用匿名类的方式来注册监听器，也可以使用实现接口的方

式来进行注册，代码如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                // 在此处添加逻辑
                break;
            default:
                break;
        }
    }
}
```

这两种写法都可以实现对按钮点击事件的监听，至于使用哪一种就全凭你的喜好了。

3.2.3　EditText

EditText 是程序用于和用户进行交互的另一个重要控件，它允许用户在控件里输入和编辑内容，并可以在程序中对这些内容进行处理。EditText 的应用场景非常普遍，在进行发短信、发微博、聊 QQ 等操作时，你不得不使用 EditText。那我们来看一看如何在界面上加入 EditText 吧，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />

</LinearLayout>
```

其实看到这里，我估计你已经总结出 Android 控件的使用规律了，用法基本上都很相似：给控件定义一个 id，再指定控件的宽度和高度，然后再适当加入一些控件特有的属性就差不多了。

所以使用 XML 来编写界面其实一点都不难，完全可以不用借助任何可视化工具来实现。现在重新运行一下程序，EditText 就已经在界面上显示出来了，并且我们是可以在里面输入内容的，如图 3.5 所示。



图 3.5　EditText 运行效果

细心的你平时应该会留意到，一些做得比较人性化的软件会在输入框里显示一些提示性的文字，然后一旦用户输入了任何内容，这些提示性的文字就会消失。这种提示功能在 Android 里是非常容易实现的，我们甚至不需要做任何的逻辑控制，因为系统已经帮我们都处理好了。修改 activity_main.xml，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Type something here"
        />

</LinearLayout>
```

这里使用 `android:hint` 属性指定了一段提示性的文本，然后重新运行程序，效果如图 3.6 所示。

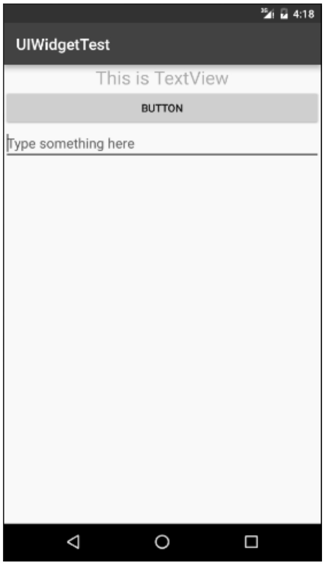


图 3.6 EditText 设置 hint 效果

可以看到，EditText 中显示了一段提示性文本，然后当我们输入任何内容时，这段文本就会自动消失。

不过，随着输入的内容不断增多，EditText 会被不断地拉长。这时由于 EditText 的高度指定的是 `wrap_content`，因此它总能包含住里面的内容，但是当输入的内容过多时，界面就会变得非常难看。我们可以使用 `android:maxLines` 属性来解决这个问题，修改 activity_main.xml，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Type something here"
        android:maxLines="2"
        />

</LinearLayout>
```

这里通过 `android:maxLines` 指定了 EditText 的最大行数为两行，这样当输入的内容超过两行时，文本就会向上滚动，而 EditText 则不会再继续拉伸，如图 3.7 所示。



图 3.7　EditText 设置 maxLines 效果

我们还可以结合使用 EditText 与 Button 来完成一些功能，比如通过点击按钮来获取 EditText 中输入的内容。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText editText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        editText = (EditText) findViewById(R.id.edit_text);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                String inputText = editText.getText().toString();
                Toast.makeText(MainActivity.this, inputText,
                Toast.LENGTH_SHORT).show();
                break;
            default:
                break;
        }
    }

}
```

首先通过 `findViewById()`方法得到 EditText 的实例，然后在按钮的点击事件里调用 EditText 的 `getText()`方法获取到输入的内容，再调用 `toString()`方法转换成字符串，最后还是老方法，使用 Toast 将输入的内容显示出来。

重新运行程序，在 EditText 中输入一段内容，然后点击按钮，效果如图 3.8 所示。

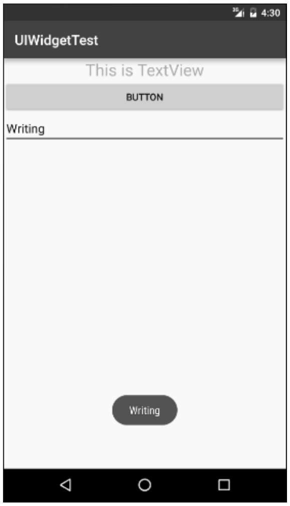


图 3.8　获取 EditText 中输入的内容

3.2.4　ImageView

ImageView 是用于在界面上展示图片的一个控件，它可以让我们的程序界面变得更加丰富多彩。学习这个控件需要提前准备好一些图片，图片通常都是放在以“drawable”开头的目录下的。目前我们的项目中有一个空的 drawable 目录，不过由于这个目录没有指定具体的分辨率，所以一般不使用它来放置图片。这里我们在 res 目录下新建一个 drawable-xhdpi 目录，然后将事先准备好的两张图片 img_1.png 和 img_2.png 复制到该目录当中。

接下来修改 activity_main.xml，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <ImageView
        android:id="@+id/image_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/img_1 "
```

```
        />

</LinearLayout>
```

可以看到，这里使用 `android:src` 属性给 ImageView 指定了一张图片。由于图片的宽和高都是未知的，所以将 ImageView 的宽和高都设定为 `wrap_content`，这样就保证了不管图片的尺寸是多少，图片都可以完整地展示出来。重新运行程序，效果如图 3.9 所示。



图 3.9 ImageView 运行效果

我们还可以在程序中通过代码动态地更改 ImageView 中的图片，然后修改 MainActivity 的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText editText;

    private ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        editText = (EditText) findViewById(R.id.edit_text);
        imageView  = (ImageView) findViewById(R.id.image_view);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                imageView.setImageResource(R.drawable.img_2);
```

```
                break;
            default:
                break;
        }
    }

}
```

在按钮的点击事件里，通过调用 ImageView 的 `setImageResource()`方法将显示的图片改成 img_2，现在重新运行程序，然后点击一下按钮，就可以看到 ImageView 中显示的图片改变了，如图 3.10 所示。

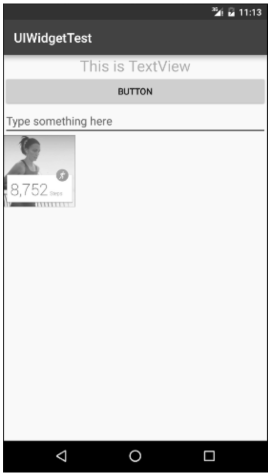


图 3.10　动态更改 ImageView 中的图片

3.2.5　ProgressBar

ProgressBar 用于在界面上显示一个进度条，表示我们的程序正在加载一些数据。它的用法也非常简单，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <ProgressBar
        android:id="@+id/progress_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />

</LinearLayout>
```

重新运行程序，会看到屏幕中有一个圆形进度条正在旋转，如图 3.11 所示。

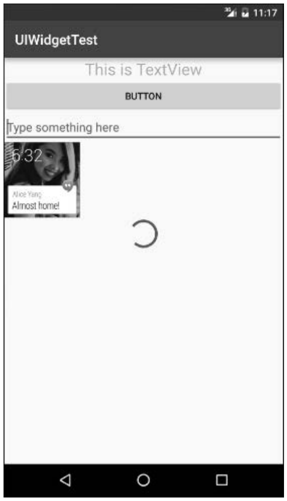


图 3.11　ProgressBar 运行效果

这时你可能会问，旋转的进度条表明我们的程序正在加载数据，那数据总会有加载完的时候吧？如何才能让进度条在数据加载完成时消失呢？这里我们就需要用到一个新的知识点：Android 控件的可见属性。所有的 Android 控件都具有这个属性，可以通过 `android:visibility` 进行指定，可选值有 3 种：`visible`、`invisible` 和 `gone`。`visible` 表示控件是可见的，这个值是默认值，不指定 `android:visibility` 时，控件都是可见的。`invisible` 表示控件不可见，但是它仍然占据着原来的位置和大小，可以理解成控件变成透明状态了。`gone` 则表示控件不仅不可见，而且不再占用任何屏幕空间。我们还可以通过代码来设置控件的可见性，使用的是 `setVisibility()`方法，可以传入 `View.VISIBLE`、`View.INVISIBLE` 和 `View.GONE` 这 3 种值。

接下来我们就来尝试实现，点击一下按钮让进度条消失，再点击一下按钮让进度条出现的这种效果。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private EditText editText;

    private ImageView imageView;

    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        editText = (EditText) findViewById(R.id.edit_text);
        imageView  = (ImageView) findViewById(R.id.image_view);
```

```
        progressBar = (ProgressBar) findViewById(R.id.progress_bar);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                if (progressBar.getVisibility() == View.GONE) {
                    progressBar.setVisibility(View.VISIBLE);
                } else {
                    progressBar.setVisibility(View.GONE);
                }
                break;
            default:
                break;
        }
    }

}
```

在按钮的点击事件中，我们通过 `getVisibility()`方法来判断 ProgressBar 是否可见，如果可见就将 ProgressBar 隐藏掉，如果不可见就将 ProgressBar 显示出来。重新运行程序，然后不断地点击按钮，你就会看到进度条会在显示与隐藏之间来回切换。

另外，我们还可以给 ProgressBar 指定不同的样式，刚刚是圆形进度条，通过 `style` 属性可以将它指定成水平进度条，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <ProgressBar
        android:id="@+id/progress_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="?android:attr/progressBarStyleHorizontal"
        android:max="100"
        />

</LinearLayout>
```

指定成水平进度条后，我们还可以通过 `android:max` 属性给进度条设置一个最大值，然后在代码中动态地更改进度条的进度。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    @Override
```

```
        public void onClick(View v) {
            switch (v.getId()) {
                case R.id.button:
                    int progress = progressBar.getProgress();
                    progress = progress + 10;
                    progressBar.setProgress(progress);
                    break;
                default:
                    break;
            }
        }

    }
```

每点击一次按钮，我们就获取进度条的当前进度，然后在现有的进度上加 10 作为更新后的进度。重新运行程序，点击数次按钮后，效果如图 3.12 所示。



图 3.12 ProgressBar 水平样式效果

ProgressBar 还有几种其他的样式，你可以自己去尝试一下。

3.2.6 AlertDialog

AlertDialog 可以在当前的界面弹出一个对话框，这个对话框是置顶于所有界面元素之上的，能够屏蔽掉其他控件的交互能力，因此 AlertDialog 一般都是用于提示一些非常重要的内容或者警告信息。比如为了防止用户误删重要内容，在删除前弹出一个确认对话框。下面我们来学习一下它的用法，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    @Override
```

```
        public void onClick(View v) {
            switch (v.getId()) {
                case R.id.button:
                    AlertDialog.Builder dialog = new AlertDialog.Builder (MainActivity.
                        this);
                    dialog.setTitle("This is Dialog");
                    dialog.setMessage("Something important.");
                    dialog.setCancelable(false);
                    dialog.setPositiveButton("OK", new DialogInterface.
                        OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                        }
                    });
                    dialog.setNegativeButton("Cancel", new DialogInterface.
                        OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                        }
                    });
                    dialog.show();
                    break;
                default:
                    break;
            }
        }
    }
```

首先通过 AlertDialog.Builder 创建一个 AlertDialog 的实例，然后可以为这个对话框设置标题、内容、可否用 Back 键关闭对话框等属性，接下来调用 `setPositiveButton()`方法为对话框设置确定按钮的点击事件，调用 `setNegativeButton()`方法设置取消按钮的点击事件，最后调用 `show()`方法将对话框显示出来。重新运行程序，点击按钮后，效果如图 3.13 所示。

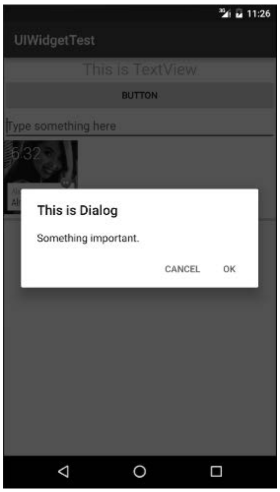


图 3.13　AlertDialog 运行效果

3.2.7　ProgressDialog

ProgressDialog 和 AlertDialog 有点类似，都可以在界面上弹出一个对话框，都能够屏蔽掉其他控件的交互能力。不同的是，ProgressDialog 会在对话框中显示一个进度条，一般用于表示当前操作比较耗时，让用户耐心地等待。它的用法和 AlertDialog 也比较相似，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                ProgressDialog progressDialog = new ProgressDialog
                    (MainActivity.this);
                progressDialog.setTitle("This is ProgressDialog");
                progressDialog.setMessage("Loading...");
                progressDialog.setCancelable(true);
                progressDialog.show();
                break;
            default:
                break;
        }
    }

}
```

可以看到，这里也是先构建出一个 `ProgressDialog` 对象，然后同样可以设置标题、内容、可否取消等属性，最后也是通过调用 `show()`方法将 ProgressDialog 显示出来。重新运行程序，点击按钮后，效果如图 3.14 所示。

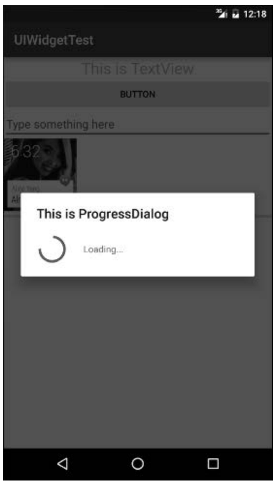


图 3.14　ProgressDialog 运行效果

注意，如果在 `setCancelable()`中传入了 `false`，表示 ProgressDialog 是不能通过 Back 键取消掉的，这时你就一定要在代码中做好控制，当数据加载完成后必须要调用 ProgressDialog 的 `dismiss()`方法来关闭对话框，否则 ProgressDialog 将会一直存在。

好了，关于 Android 常用控件的使用，我要讲的就只有这么多。一节内容就想覆盖 Android 控件所有的相关知识不太现实，同样一口气就想学会所有 Android 控件的使用方法也不太现实。本节所讲的内容对于你来说只是起到了一个引导的作用，你还需要在以后的学习和工作中不断地摸索，通过查阅文档以及网上搜索的方式学习更多控件的更多用法。当然，当本书后面涉及一些我们前面没学过的控件和相关用法时，我仍然会在相应的章节做详细的讲解。

3.3　详解 4 种基本布局

一个丰富的界面总是要由很多个控件组成的，那我们如何才能让各个控件都有条不紊地摆放在界面上，而不是乱糟糟的呢？这就需要借助布局来实现了。布局是一种可用于放置很多控件的容器，它可以按照一定的规律调整内部控件的位置，从而编写出精美的界面。当然，布局的内部除了放置控件外，也可以放置布局，通过多层布局的嵌套，我们就能够完成一些比较复杂的界面实现，图 3.15 很好地展示了它们之间的关系。

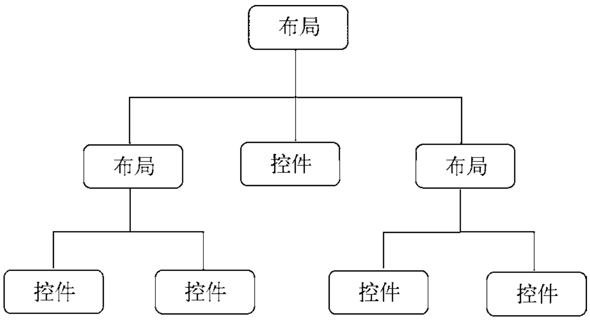


图 3.15　布局和控件的关系

下面我们来详细讲解下 Android 中 4 种最基本的布局。先做好准备工作，新建一个 UILayoutTest 项目，并让 Android Studio 自动帮我们创建好活动，活动名和布局名都使用默认值。

3.3.1　线性布局

LinearLayout 又称作线性布局，是一种非常常用的布局。正如它的名字所描述的一样，这个布局会将它所包含的控件在线性方向上依次排列。相信你之前也已经注意到了，我们在上一节中学习控件用法时，所有的控件就都是放在 LinearLayout 布局里的，因此上一节中的控件也确实是在垂直方向上线性排列的。

既然是线性排列，肯定就不仅只有一个方向，那为什么上一节中的控件都是在垂直方向排列

的呢？这是由于我们通过 `android:orientation` 属性指定了排列方向是 vertical，如果指定的是 horizontal，控件就会在水平方向上排列了。下面我们通过实战来体会一下，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3" />

</LinearLayout>
```

我们在 LinearLayout 中添加了 3 个 Button，每个 Button 的长和宽都是 `wrap_content`，并指定了排列方向是 vertical。现在运行一下程序，效果如图 3.16 所示。

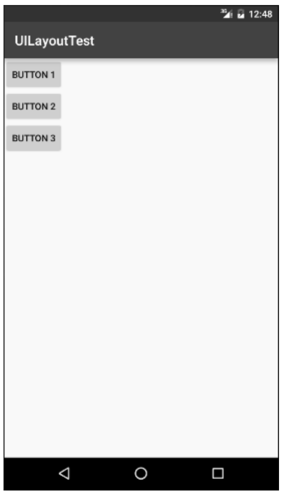


图 3.16 LinearLayout 垂直排列

然后我们修改一下 LinearLayout 的排列方向，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

</LinearLayout>
```

将 `android:orientation` 属性的值改成了 horizontal，这就意味着要让 LinearLayout 中的控件在水平方向上依次排列。当然如果不指定 `android:orientation` 属性的值，默认的排列方向就是 horizontal。重新运行一下程序，效果如图 3.17 所示。

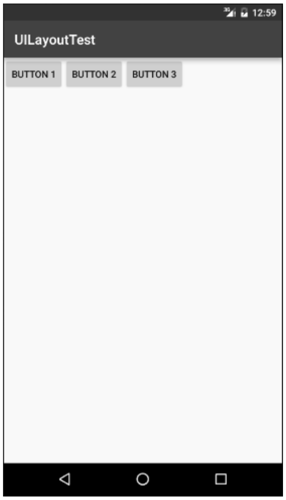


图 3.17　LinearLayout 水平排列

这里需要注意，如果 LinearLayout 的排列方向是 horizontal，内部的控件就绝对不能将宽度指定为 `match_parent`，因为这样的话，单独一个控件就会将整个水平方向占满，其他的控件就没有可放置的位置了。同样的道理，如果 LinearLayout 的排列方向是 vertical，内部的控件就不能将高度指定为 `match_parent`。

首先来看 `android:layout_gravity` 属性，它和我们上一节中学到的 `android:gravity` 属性看起来有些相似，这两个属性有什么区别呢？其实从名字就可以看出，`android:gravity` 用于指定文字在控件中的对齐方式，而 `android:layout_gravity` 用于指定控件在布局中的对齐方式。`android:layout_gravity` 的可选值和 `android:gravity` 差不多，但是需要注意，当 LinearLayout 的排列方向是 horizontal 时，只有垂直方向上的对齐方式才会生效，因为此时水平方向上的长度是不固定的，每添加一个控件，水平方向上的长度都会改变，因而无法指定该方向上的对齐方式。同样的道理，当 LinearLayout 的排列方向是 vertical 时，只有水平方向上的对齐方

式才会生效。修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:text="Button 3" />

</LinearLayout>
```

由于目前 LinearLayout 的排列方向是 horizontal，因此我们只能指定垂直方向上的排列方向，将第一个 Button 的对齐方式指定为 top，第二个 Button 的对齐方式指定为 center_vertical，第三个 Button 的对齐方式指定为 bottom。重新运行程序，效果如图 3.18 所示。

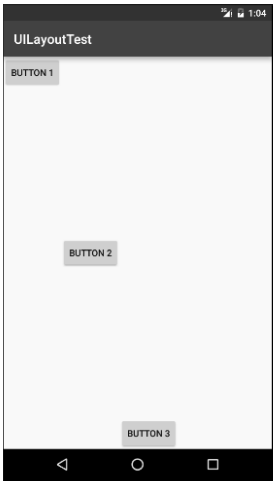


图 3.18　指定 layout_gravity 的效果

接下来我们学习下 LinearLayout 中的另一个重要属性——`android:layout_weight`。这个属性允许我们使用比例的方式来指定控件的大小，它在手机屏幕的适配性方面可以起到非常重要的作用。比如我们正在编写一个消息发送界面，需要一个文本编辑框和一个发送按钮，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/input_message"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="Type something"
        />

    <Button
        android:id="@+id/send"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Send"
        />

</LinearLayout>
```

你会发现，这里竟然将 EditText 和 Button 的宽度都指定成了 0dp，这样文本编辑框和按钮还能显示出来吗？不用担心，由于我们使用了 `android:layout_weight` 属性，此时控件的宽度就不应该再由 `android:layout_width` 来决定，这里指定成 0dp 是一种比较规范的写法。另外，dp 是 Android 中用于指定控件大小、间距等属性的单位，后面我们还会经常用到它。

然后在 EditText 和 Button 里都将 `android:layout_weight` 属性的值指定为 1，这表示 EditText 和 Button 将在水平方向平分宽度。

为什么将 `android:layout_weight` 属性的值同时指定为 1 就会平分屏幕宽度呢？其实原理也很简单，系统会先把 LinearLayout 下所有控件指定的 `layout_weight` 值相加，得到一个总值，然后每个控件所占大小的比例就是用该控件的 `layout_weight` 值除以刚才算出的总值。因此如果想让 EditText 占据屏幕宽度的 3/5，Button 占据屏幕宽度的 2/5，只需要将 EditText 的 `layout_weight` 改成 3，Button 的 `layout_weight` 改成 2 就可以了。

重新运行程序，你会看到如图 3.19 所示的效果。

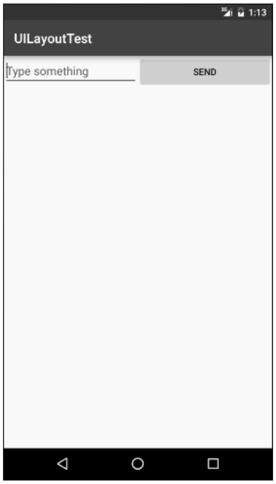


图 3.19 指定 layout_weight 的效果

我们还可以通过指定部分控件的 layout_weight 值来实现更好的效果。修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/input_message"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="Type something"
        />

    <Button
        android:id="@+id/send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send"
        />

</LinearLayout>
```

这里我们仅指定了 EditText 的 android:layout_weight 属性，并将 Button 的宽度改回 wrap_content。这表示 Button 的宽度仍然按照 wrap_content 来计算，而 EditText 则会占满屏幕所有的剩余空间。使用这种方式编写的界面，不仅在各种屏幕的适配方面会非常好，而且看起来也更加舒服。重新运行程序，效果如图 3.20 所示。

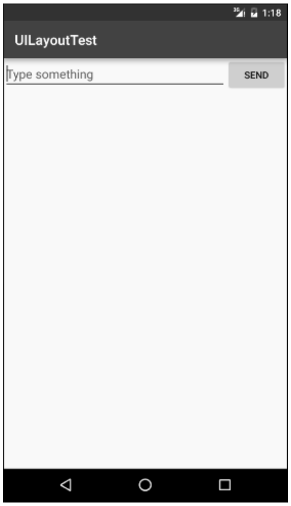


图 3.20　使用 `layout_weight` 实现宽度自适配效果

3.3.2　相对布局

RelativeLayout 又称作相对布局，也是一种非常常用的布局。和 LinearLayout 的排列规则不同，RelativeLayout 显得更加随意一些，它可以通过相对定位的方式让控件出现在布局的任何位置。也正因为如此，RelativeLayout 中的属性非常多，不过这些属性都是有规律可循的，其实并不难理解和记忆。我们还是通过实践来体会一下，修改 activity_main.xml 中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Button 3" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:text="Button 4" />

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:text="Button 5" />

</RelativeLayout>
```

我想以上代码已经不需要我再做过多解释了，因为实在是太好理解了。我们让 Button 1 和父布局的左上角对齐，Button 2 和父布局的右上角对齐，Button 3 居中显示，Button 4 和父布局的左下角对齐，Button 5 和父布局的右下角对齐。虽然 `android:layout_alignParentLeft`、`android:layout_alignParentTop`、`android:layout_alignParentRight`、`android:layout_alignParentBottom`、`android:layout_centerInParent` 这几个属性我们之前都没接触过，可是它们的名字已经完全说明了它们的作用。重新运行程序，效果如图 3.21 所示。

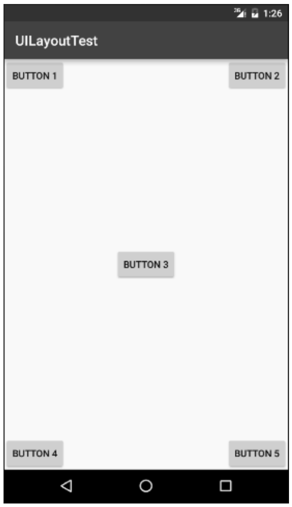


图 3.21 相对于父布局定位的效果

上面例子中的每个控件都是相对于父布局进行定位的，那控件可不可以相对于控件进行定位呢？当然是可以的，修改 activity_main.xml 中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Button 3" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/button3"
        android:layout_toLeftOf="@id/button3"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/button3"
        android:layout_toRightOf="@id/button3"
        android:text="Button 2" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button3"
        android:layout_toLeftOf="@id/button3"
        android:text="Button 4" />

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button3"
        android:layout_toRightOf="@id/button3"
        android:text="Button 5" />

</RelativeLayout>
```

这次的代码稍微复杂一点，不过仍然是有规律可循的。`android:layout_above` 属性可以让一个控件位于另一个控件的上方，需要为这个属性指定相对控件 id 的引用，这里我们填入了 `@id/button3`，表示让该控件位于 Button 3 的上方。其他的属性也都是相似的，`android:layout_below` 表示让一个控件位于另一个控件的下方，`android:layout_toLeftOf` 表示让一

个控件位于另一个控件的左侧，`android:layout_toRightOf` 表示让一个控件位于另一个控件的右侧。注意，当一个控件去引用另一个控件的 id 时，该控件一定要定义在引用控件的后面，不然会出现找不到 id 的情况。重新运行程序，效果如图 3.22 所示。

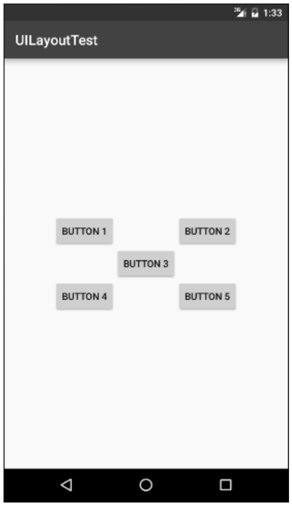


图 3.22 相对于控件定位的效果

RelativeLayout 中还有另外一组相对于控件进行定位的属性，`android:layout_alignLeft` 表示让一个控件的左边缘和另一个控件的左边缘对齐，`android:layout_alignRight` 表示让一个控件的右边缘和另一个控件的右边缘对齐。此外，还有 `android:layout_alignTop` 和 `android:layout_alignBottom`，道理都是一样的，我就不再多说，这几个属性就留给你自己去尝试吧。

好了，正如我前面所说，RelativeLayout 中的属性虽然多，但都是有规律可循的，所以学起来一点都不觉得吃力吧？

3.3.3 帧布局

FrameLayout 又称作帧布局，它相比于前面两种布局就简单太多了，因此它的应用场景也少了很多。这种布局没有方便的定位方式，所有的控件都会默认摆放在布局的左上角。让我们通过例子来看一看吧，修改 activity_main.xml 中的代码，如下所示：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="This is TextView"
            />

        <ImageView
            android:id="@+id/image_view"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/ic_launcher"
            />

    </FrameLayout>
```

FrameLayout 中只是放置了一个 TextView 和一个 ImageView。需要注意的是，当前项目我们没有准备任何图片，所以这里 ImageView 直接使用了`@mipmap` 来访问 ic_launcher 这张图，虽说这种用法的场景可能非常少，但我还是要告诉你，这是完全可行的。重新运行程序，效果如图 3.23 所示。

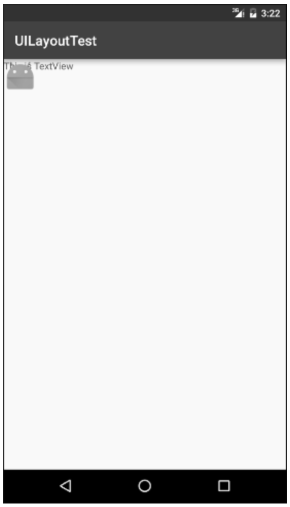


图 3.23　FrameLayout 运行效果

可以看到，文字和图片都是位于布局的左上角。由于 ImageView 是在 TextView 之后添加的，因此图片压在了文字的上面。

当然除了这种默认效果之外，我们还可以使用 `layout_gravity` 属性来指定控件在布局中的对齐方式，这和 LinearLayout 中的用法是相似的。修改 activity_main.xml 中的代码，如下所示：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text_view"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:text="This is TextView"
        />

    <ImageView
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:src="@mipmap/ic_launcher"
        />

</FrameLayout>
```

我们指定 TextView 在 FrameLayout 中居左对齐，指定 ImageView 在 FrameLayout 中居右对齐，然后重新运行程序，效果如图 3.24 所示。

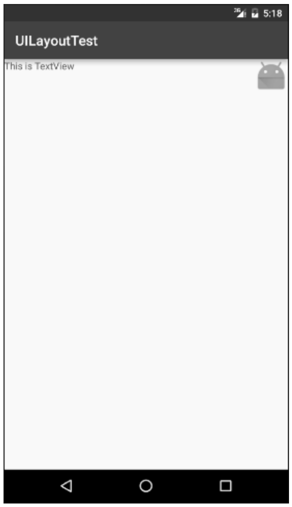


图 3.24 指定 `layout_gravity` 的效果

总体来讲，FrameLayout 由于定位方式的欠缺，导致它的应用场景也比较少，不过在下一章中介绍碎片的时候我们还是可以用到它的。

3.3.4 百分比布局

前面介绍的 3 种布局都是从 Android 1.0 版本中就开始支持了，一直沿用到现在，可以说是满足了绝大多数场景的界面设计需求。不过细心的你会发现，只有 LinearLayout 支持使用 `layout_weight` 属性来实现按比例指定控件大小的功能，其他两种布局都不支持。比如说，如果想用 RelativeLayout 来实现让两个按钮平分布局宽度的效果，则是比较困难的。

为此，Android 引入了一种全新的布局方式来解决此问题——百分比布局。在这种布局中，我们可以不再使用 `wrap_content`、`match_parent` 等方式来指定控件的大小，而是允许直接指定控件在布局中所占的百分比，这样的话就可以轻松实现平分布局甚至是任意比例分割布局的效果了。

由于 LinearLayout 本身已经支持按比例指定控件的大小了，因此百分比布局只为 FrameLayout 和 RelativeLayout 进行了功能扩展，提供了 PercentFrameLayout 和 PercentRelativeLayout 这两个全新的布局，下面我们就来具体学习一下。

不同于前 3 种布局，百分比布局属于新增布局，那么怎么才能做到让新增布局在所有 Android 版本上都能使用呢？为此，Android 团队将百分比布局定义在了 support 库当中，我们只需要在项目的 build.gradle 中添加百分比布局库的依赖，就能保证百分比布局在 Android 所有系统版本上的兼容性了。

打开 app/build.gradle 文件，在 `dependencies` 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:percent:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

需要注意的是，每当修改了任何 gradle 文件时，Android Studio 都会弹出一个如图 3.25 所示的提示。

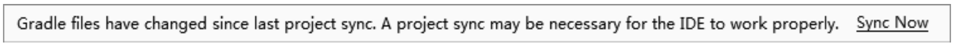


图 3.25　gradle 文件修改后的提示

这个提示告诉我们，gradle 文件自上次同步之后又发生了变化，需要再次同步才能使项目正常工作。这里只需要点击 Sync Now 就可以了，然后 gradle 会开始进行同步，把我们新添加的百分比布局库引入到项目当中。

接下来修改 activity_main.xml 中的代码，如下所示：

```
<android.support.percent.PercentFrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button1"
        android:text="Button 1"
        android:layout_gravity="left|top"
        app:layout_widthPercent="50%"
        app:layout_heightPercent="50%"
```

```
            />

        <Button
            android:id="@+id/button2"
            android:text="Button 2"
            android:layout_gravity="right|top"
            app:layout_widthPercent="50%"
            app:layout_heightPercent="50%"
            />

        <Button
            android:id="@+id/button3"
            android:text="Button 3"
            android:layout_gravity="left|bottom"
            app:layout_widthPercent="50%"
            app:layout_heightPercent="50%"
            />

        <Button
            android:id="@+id/button4"
            android:text="Button 4"
            android:layout_gravity="right|bottom"
            app:layout_widthPercent="50%"
            app:layout_heightPercent="50%"
            />

    </android.support.percent.PercentFrameLayout>
```

最外层我们使用了 PercentFrameLayout，由于百分比布局并不是内置在系统 SDK 当中的，所以需要把完整的包路径写出来。然后还必须定义一个 app 的命名空间，这样才能使用百分比布局的自定义属性。

在 PercentFrameLayout 中我们定义了 4 个按钮，使用 `app:layout_widthPercent` 属性将各按钮的宽度指定为布局的 50%，使用 `app:layout_heightPercent` 属性将各按钮的高度指定为布局的 50%。这里之所以能使用 app 前缀的属性就是因为刚才定义了 app 的命名空间，当然我们一直能使用 android 前缀的属性也是同样的道理。

不过 PercentFrameLayout 还是会继承 FrameLayout 的特性，即所有的控件默认都是摆放在布局的左上角。那么为了让这 4 个按钮不会重叠，这里还是借助了 `layout_gravity` 来分别将这 4 个按钮放置在布局的左上、右上、左下、右下 4 个位置。

现在我们已经可以重新运行程序了，不过如果你使用的是老版本的 Android Studio，可能会在 activity_main.xml 中看到一些如图 3.26 所示的错误提示。

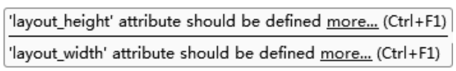


图 3.26　activity_main.xml 中错误提示

这是因为老版本的 Android Studio 中内置了布局的检查机制，认为每一个控件都应该通过 `android:layout_width` 和 `android:layout_height` 属性指定宽高才是合法的。而其实我们是通过 `app:layout_widthPercent` 和 `app:layout_heightPercent` 属性来指定宽高的，所以 Android Studio 没检测到。不过这个错误提示并不影响程序运行，我们直接忽视就可以了。当然最新的 Android Studio 2.2 版本中已经修复了这个问题，因此你可能并不会看到上述的错误提示。

现在重新运行程序，效果如图 3.27 所示。

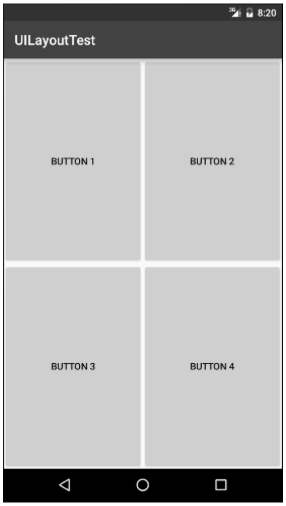


图 3.27　PercentFrameLayout 运行效果

可以看到，每一个按钮的宽和高都占据了布局的 50%，这样我们就轻松实现了 4 个按钮平分屏幕的效果。

PercentFrameLayout 的用法就介绍到这里，另外一个 PercentRelativeLayout 的用法也是非常相似的，它继承了 RelativeLayout 中的所有属性，并且可以使用 `app:layout_widthPercent` 和 `app:layout_heightPercent` 来按百分比指定控件的宽高，相信聪明的你一定可以举一反三了。

这样我们就把最常用的几种布局都讲解完了，其实 Android 中还有 AbsoluteLayout、TableLayout 等布局，不过由于使用得实在是太少了，就不在本书中进行讲解了。

3.4　系统控件不够用？创建自定义控件

在前面两节我们已经学习了 Android 中的一些常用控件以及基本布局的用法，不过当时我们并没有关注这些控件和布局的继承结构，现在是时候来看一下了，如图 3.28 所示。

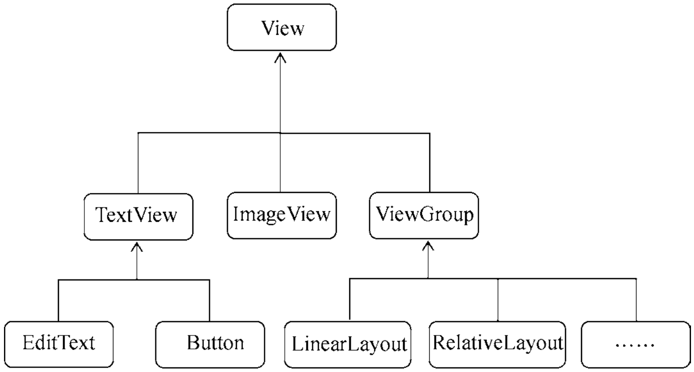


图 3.28　常用控件和布局的继承结构

可以看到，我们所用的所有控件都是直接或间接继承自 View 的，所用的所有布局都是直接或间接继承自 ViewGroup 的。View 是 Android 中最基本的一种 UI 组件，它可以在屏幕上绘制一块矩形区域，并能响应这块区域的各种事件，因此，我们使用的各种控件其实就是在 View 的基础之上又添加了各自特有的功能。而 ViewGroup 则是一种特殊的 View，它可以包含很多子 View 和子 ViewGroup，是一个用于放置控件和布局的容器。

这个时候我们就可以思考一下，当系统自带的控件并不能满足我们的需求时，可不可以利用上面的继承结构来创建自定义控件呢？答案是肯定的，下面我们就来学习一下创建自定义控件的两种简单方法。先将准备工作做好，创建一个 UICustomViews 项目。

3.4.1　引入布局

如果你用过 iPhone 应该会知道，几乎每一个 iPhone 应用的界面顶部都会有一个标题栏，标题栏上会有一到两个按钮可用于返回或其他操作（iPhone 没有实体返回键）。现在很多 Android 程序也都喜欢模仿 iPhone 的风格，在界面的顶部放置一个标题栏。虽然 Android 系统已经给每个活动提供了标题栏功能，但这里我们决定先不使用它，而是创建一个自定义的标题栏。

经过前面两节的学习，相信创建一个标题栏布局对你来说已经不是什么困难的事情了，只需要加入两个 Button 和一个 TextView，然后在布局中摆放好就可以了。可是这样做却存在着一个问题，一般我们的程序中可能有很多个活动都需要这样的标题栏，如果在每个活动的布局中都编写一遍同样的标题栏代码，明显就会导致代码的大量重复。这个时候我们就可以使用引入布局的方式来解决这个问题，新建一个布局 title.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/title_bg">

    <Button
        android:id="@+id/title_back"
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="5dp"
            android:background="@drawable/back_bg"
            android:text="Back"
            android:textColor="#fff" />

        <TextView
            android:id="@+id/title_text"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_weight="1"
            android:gravity="center"
            android:text="Title Text"
            android:textColor="#fff"
            android:textSize="24sp" />

        <Button
            android:id="@+id/title_edit"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="5dp"
            android:background="@drawable/edit_bg"
            android:text="Edit"
            android:textColor="#fff" />

    </LinearLayout>
```

可以看到，我们在 LinearLayout 中分别加入了两个 Button 和一个 TextView，左边的 Button 可用于返回，右边的 Button 可用于编辑，中间的 TextView 则可以显示一段标题文本。上面代码中的大多数属性都是你已经见过的，下面我来说明一下几个之前没有讲过的属性。`android:background` 用于为布局或控件指定一个背景，可以使用颜色或图片来进行填充，这里我提前准备好了 3 张图片——title_bg.png、back_bg.png 和 edit_bg.png，分别用于作为标题栏、返回按钮和编辑按钮的背景。另外，在两个 Button 中我们都使用了 `android:layout_margin` 这个属性，它可以指定控件在上下左右方向上偏移的距离，当然也可以使用 `android:layout_marginLeft` 或 `android:layout_marginTop` 等属性来单独指定控件在某个方向上偏移的距离。

现在标题栏布局已经编写完成了，剩下的就是如何在程序中使用这个标题栏了，修改 activity_main.xml 中的代码，如下所示：

```
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <include layout="@layout/title" />

    </LinearLayout>
```

没错！我们只需要通过一行 `include` 语句将标题栏布局引入进来就可以了。

最后别忘了在 MainActivity 中将系统自带的标题栏隐藏掉，代码如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ActionBar actionbar = getSupportActionBar();
        if (actionbar != null) {
            actionbar.hide();
        }
    }

}
```

这里我们调用了 `getSupportActionBar()`方法来获得 ActionBar 的实例，然后再调用 ActionBar 的 `hide()`方法将标题栏隐藏起来。关于 ActionBar 的更多用法我们将会在第 12 章中讲解，现在你只需要知道可以通过这种写法来隐藏标题栏就足够了。现在运行一下程序，效果如图 3.29 所示。

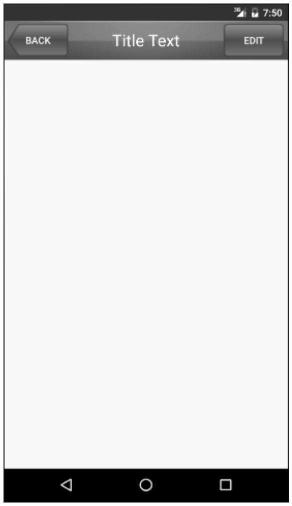


图 3.29　引入标题栏布局的效果

使用这种方式，不管有多少布局需要添加标题栏，只需一行 `include` 语句就可以了。

3.4.2　创建自定义控件

引入布局的技巧确实解决了重复编写布局代码的问题，但是如果布局中有一些控件要求能够响应事件，我们还是需要在每个活动中为这些控件单独编写一次事件注册的代码。比如说标题栏中的返回按钮，其实不管是在哪一个活动中，这个按钮的功能都是相同的，即销毁当前活动。而

如果在每一个活动中都需要重新注册一遍返回按钮的点击事件，无疑会增加很多重复代码，这种情况最好是使用自定义控件的方式来解决。

新建 TitleLayout 继承自 LinearLayout，让它成为我们自定义的标题栏控件，代码如下所示：

```
public class TitleLayout extends LinearLayout {

    public TitleLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        LayoutInflater.from(context).inflate(R.layout.title, this);
    }

}
```

首先我们重写了 LinearLayout 中带有两个参数的构造函数，在布局中引入 TitleLayout 控件就会调用这个构造函数。然后在构造函数中需要对标题栏布局进行动态加载，这就要借助 LayoutInflater 来实现了。通过 LayoutInflater 的 `from()`方法可以构建出一个 `LayoutInflater` 对象，然后调用 `inflate()`方法就可以动态加载一个布局文件，`inflate()`方法接收两个参数，第一个参数是要加载的布局文件的 id，这里我们传入 R.layout.title，第二个参数是给加载好的布局再添加一个父布局，这里我们想要指定为 TitleLayout，于是直接传入 `this`。

现在自定义控件已经创建好了，然后我们需要在布局文件中添加这个自定义控件，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <com.example.uicustomviews.TitleLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

添加自定义控件和添加普通控件的方式基本是一样的，只不过在添加自定义控件的时候，我们需要指明控件的完整类名，包名在这里是不可以省略的。

重新运行程序，你会发现此时效果和使用引入布局方式的效果是一样的。

下面我们尝试为标题栏中的按钮注册点击事件，修改 TitleLayout 中的代码，如下所示：

```
public class TitleLayout extends LinearLayout {

    public TitleLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        LayoutInflater.from(context).inflate(R.layout.title, this);
        Button titleBack = (Button) findViewById(R.id.title_back);
        Button titleEdit = (Button) findViewById(R.id.title_edit);
        titleBack.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                ((Activity) getContext()).finish();
            }
        });
        titleEdit.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getContext(), "You clicked Edit button",
                    Toast.LENGTH_SHORT).show();
            }
        });
    }

}
```

首先还是通过findViewById()方法得到按钮的实例,然后分别调用setOnClickListener()方法给两个按钮注册了点击事件，当点击返回按钮时销毁掉当前的活动，当点击编辑按钮时弹出一段文本。重新运行程序，点击一下编辑按钮，效果如图 3.30 所示。



图 3.30 点击编辑按钮的效果

这样的话，每当我们在一个布局中引入 TitleLayout 时，返回按钮和编辑按钮的点击事件就已经自动实现好了，这就省去了很多编写重复代码的工作。

3.5 最常用和最难用的控件——ListView

ListView 绝对可以称得上是 Android 中最常用的控件之一，几乎所有的应用程序都会用到它。由于手机屏幕空间都比较有限，能够一次性在屏幕上显示的内容并不多，当我们的程序中有大量的数据需要展示的时候，就可以借助 ListView 来实现。ListView 允许用户通过手指上下滑动的方式将屏幕外的数据滚动到屏幕内，同时屏幕上原有的数据则会滚动出屏幕。相信你其实每天都在使用这个控件，比如查看 QQ 聊天记录，翻阅微博最新消息，等等。

不过比起前面介绍的几种控件，ListView 的用法也相对复杂了很多，因此我们就单独使用一节内容来对 ListView 进行非常详细的讲解。

3.5.1　ListView 的简单用法

首先新建一个 ListViewTest 项目，并让 Android Studio 自动帮我们创建好活动。然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

在布局中加入 ListView 控件还算非常简单，先为 ListView 指定一个 id，然后将宽度和高度都设置为 `match_parent`，这样 ListView 也就占满了整个布局的空间。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private String[] data = { "Apple", "Banana", "Orange", "Watermelon",
        "Pear", "Grape", "Pineapple", "Strawberry", "Cherry", "Mango",
        "Apple", "Banana", "Orange", "Watermelon", "Pear", "Grape",
        "Pineapple", "Strawberry", "Cherry", "Mango" };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            MainActivity.this, android.R.layout.simple_list_item_1, data);
        ListView listView = (ListView) findViewById(R.id.list_view);
        listView.setAdapter(adapter);
    }

}
```

既然 ListView 是用于展示大量数据的，那我们就应该先将数据提供好。这些数据可以是从网上下载的，也可以是从数据库中读取的，应该视具体的应用程序场景而定。这里我们就简单使用了一个 data 数组来测试，里面包含了很多水果的名称。

不过，数组中的数据是无法直接传递给 ListView 的，我们还需要借助适配器来完成。Android 中提供了很多适配器的实现类，其中我认为最好用的就是 ArrayAdapter。它可以通过泛型来指定要适配的数据类型，然后在构造函数中把要适配的数据传入。ArrayAdapter 有多个构造函数的重

载，你应该根据实际情况选择最合适的一种。这里由于我们提供的数据都是字符串，因此将ArrayAdapter的泛型指定为`String`，然后在ArrayAdapter的构造函数中依次传入当前上下文、ListView子项布局的id，以及要适配的数据。注意，我们使用了`android.R.layout.simple_list_item_1`作为ListView子项布局的id，这是一个Android内置的布局文件，里面只有一个TextView，可用于简单地显示一段文本。这样适配器对象就构建好了。

最后，还需要调用ListView的`setAdapter()`方法，将构建好的适配器对象传递进去，这样ListView和数据之间的关联就建立完成了。

现在运行一下程序，效果如图3.31所示。可以通过滚动的方式来查看屏幕外的数据。

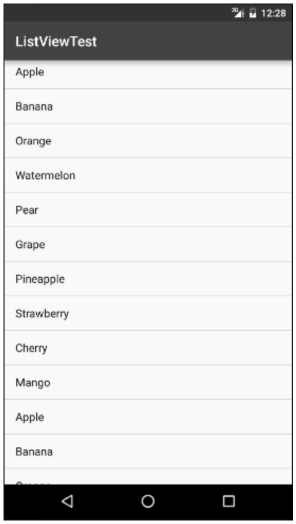


图3.31　ListView运行效果

3.5.2　定制ListView的界面

只能显示一段文本的ListView实在是太单调了，我们现在就来对ListView的界面进行定制，让它可以显示更加丰富的内容。

首先需要准备好一组图片，分别对应上面提供的每一种水果，待会我们要让这些水果名称的旁边都有一个图样。

接着定义一个实体类，作为ListView适配器的适配类型。新建类`Fruit`，代码如下所示：

```
public class Fruit {

    private String name;

    private int imageId;
```

```
    public Fruit(String name, int imageId) {
        this.name = name;
        this.imageId = imageId;
    }

    public String getName() {
        return name;
    }

    public int getImageId() {
        return imageId;
    }

}
```

Fruit 类中只有两个字段，name 表示水果的名字，imageId 表示水果对应图片的资源 id。

然后需要为 ListView 的子项指定一个我们自定义的布局，在 layout 目录下新建 fruit_item.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/fruit_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/fruit_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_marginLeft="10dp" />

</LinearLayout>
```

在这个布局中，我们定义了一个 ImageView 用于显示水果的图片，又定义了一个 TextView 用于显示水果的名称，并让 TextView 在垂直方向上居中显示。

接下来需要创建一个自定义的适配器，这个适配器继承自 ArrayAdapter，并将泛型指定为 Fruit 类。新建类 FruitAdapter，代码如下所示：

```
public class FruitAdapter extends ArrayAdapter<Fruit> {

    private int resourceId;

    public FruitAdapter(Context context, int textViewResourceId,
                        List<Fruit> objects) {
        super(context, textViewResourceId, objects);
        resourceId = textViewResourceId;
    }
```

```
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Fruit fruit = getItem(position); // 获取当前项的Fruit实例
        View view = LayoutInflater.from(getContext()).inflate(resourceId, parent,
            false);
        ImageView fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
        TextView fruitName = (TextView) view.findViewById(R.id.fruit_name);
        fruitImage.setImageResource(fruit.getImageId());
        fruitName.setText(fruit.getName());
        return view;
    }

}
```

FruitAdapter 重写了父类的一组构造函数，用于将上下文、ListView 子项布局的 id 和数据都传递进来。另外又重写了 getView()方法，这个方法在每个子项被滚动到屏幕内的时候会被调用。在 getView()方法中，首先通过 getItem()方法得到当前项的 Fruit 实例，然后使用 LayoutInflater 来为这个子项加载我们传入的布局。

这里 LayoutInflater 的 inflate()方法接收 3 个参数，前两个参数我们已经知道是什么意思了，第三个参数指定成 false，表示只让我们在父布局中声明的 layout 属性生效，但不会为这个 View 添加父布局，因为一旦 View 有了父布局之后，它就不能再添加到 ListView 中了。如果你现在还不能理解这段话的含义也没关系，只需要知道这是 ListView 中的标准写法就可以了，当你以后对 View 理解得更加深刻的时候，再来读这段话就没有问题了。

我们继续往下看，接下来调用 View 的 findViewById()方法分别获取到 ImageView 和 TextView 的实例，并分别调用它们的 setImageResource()和 setText()方法来设置显示的图片和文字，最后将布局返回，这样我们自定义的适配器就完成了。

下面修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits(); // 初始化水果数据
        FruitAdapter adapter = new FruitAdapter(MainActivity.this,
            R.layout.fruit_item, fruitList);
        ListView listView = (ListView) findViewById(R.id.list_view);
        listView.setAdapter(adapter);
    }

    private void initFruits() {
        for (int i = 0; i < 2; i++) {
            Fruit apple = new Fruit("Apple", R.drawable.apple_pic);
            fruitList.add(apple);
```

```
            Fruit banana = new Fruit("Banana", R.drawable.banana_pic);
            fruitList.add(banana);
            Fruit orange = new Fruit("Orange", R.drawable.orange_pic);
            fruitList.add(orange);
            Fruit watermelon = new Fruit("Watermelon", R.drawable.watermelon_pic);
            fruitList.add(watermelon);
            Fruit pear = new Fruit("Pear", R.drawable.pear_pic);
            fruitList.add(pear);
            Fruit grape = new Fruit("Grape", R.drawable.grape_pic);
            fruitList.add(grape);
            Fruit pineapple = new Fruit("Pineapple", R.drawable.pineapple_pic);
            fruitList.add(pineapple);
            Fruit strawberry = new Fruit("Strawberry", R.drawable.strawberry_pic);
            fruitList.add(strawberry);
            Fruit cherry = new Fruit("Cherry", R.drawable.cherry_pic);
            fruitList.add(cherry);
            Fruit mango = new Fruit("Mango", R.drawable.mango_pic);
            fruitList.add(mango);
        }
    }

}
```

可以看到，这里添加了一个 initFruits()方法，用于初始化所有的水果数据。在 Fruit 类的构造函数中将水果的名字和对应的图片 id 传入，然后把创建好的对象添加到水果列表中。另外我们使用了一个 for 循环将所有的水果数据添加了两遍，这是因为如果只添加一遍的话，数据量还不足以充满整个屏幕。接着在 onCreate()方法中创建了 FruitAdapter 对象，并将 FruitAdapter 作为适配器传递给 ListView，这样定制 ListView 界面的任务就完成了。

现在重新运行程序，效果如图 3.32 所示。

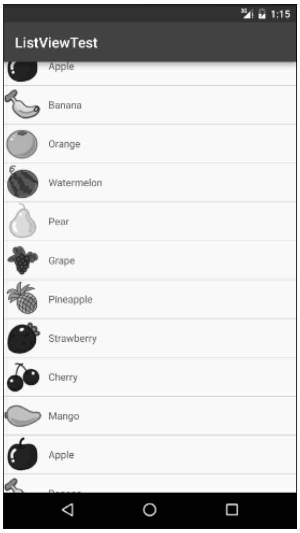


图 3.32　定制界面的 ListView 运行效果

虽然目前我们定制的界面还很简单，但是相信聪明的你已经领悟到了诀窍，只要修改 fruit_item.xml 中的内容，就可以定制出各种复杂的界面了。

3.5.3　提升 ListView 的运行效率

之所以说 ListView 这个控件很难用，就是因为它有很多细节可以优化，其中运行效率就是很重要的一点。目前我们 ListView 的运行效率是很低的，因为在 FruitAdapter 的 getView()方法中，每次都将布局重新加载了一遍，当 ListView 快速滚动的时候，这就会成为性能的瓶颈。

仔细观察会发现，getView()方法中还有一个 convertView 参数，这个参数用于将之前加载好的布局进行缓存，以便之后可以进行重用。修改 FruitAdapter 中的代码，如下所示：

```
public class FruitAdapter extends ArrayAdapter<Fruit> {

    ...

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Fruit fruit = getItem(position);
        View view;
        if (convertView == null) {
            view = LayoutInflater.from(getContext()).inflate(resourceId, parent,
                false);
        } else {
            view = convertView;
        }
        ImageView fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
        TextView fruitName = (TextView) view.findViewById(R.id.fruit_name);
        fruitImage.setImageResource(fruit.getImageId());
        fruitName.setText(fruit.getName());
        return view;
    }

}
```

可以看到，现在我们在 getView()方法中进行了判断，如果 convertView 为 null，则使用 LayoutInflater 去加载布局，如果不为 null 则直接对 convertView 进行重用。这样就大大提高了 ListView 的运行效率，在快速滚动的时候也可以表现出更好的性能。

不过，目前我们的这份代码还是可以继续优化的，虽然现在已经不会再重复去加载布局，但是每次在 getView()方法中还是会调用 View 的 findViewById()方法来获取一次控件的实例。我们可以借助一个 ViewHolder 来对这部分性能进行优化，修改 FruitAdapter 中的代码，如下所示：

```
public class FruitAdapter extends ArrayAdapter<Fruit> {

    ...

    @Override
```

```
    public View getView(int position, View convertView, ViewGroup parent) {
        Fruit fruit = getItem(position);
        View view;
        ViewHolder viewHolder;
        if (convertView == null) {
            view = LayoutInflater.from(getContext()).inflate(resourceId, parent,
            false);
            viewHolder = new ViewHolder();
            viewHolder.fruitImage = (ImageView) view.findViewById
            (R.id.fruit_image);
            viewHolder.fruitName = (TextView) view.findViewById (R.id.fruit_name);
            view.setTag(viewHolder); // 将ViewHolder存储在View中
        } else {
            view = convertView;
            viewHolder = (ViewHolder) view.getTag(); // 重新获取ViewHolder
        }
        viewHolder.fruitImage.setImageResource(fruit.getImageId());
        viewHolder.fruitName.setText(fruit.getName());
        return view;
    }

    class ViewHolder {

        ImageView fruitImage;

        TextView fruitName;

    }

}
```

我们新增了一个内部类 `ViewHolder`，用于对控件的实例进行缓存。当 `convertView` 为 `null` 的时候，创建一个 `ViewHolder` 对象，并将控件的实例都存放在 `ViewHolder` 里，然后调用 `View` 的 `setTag()`方法，将 `ViewHolder` 对象存储在 `View` 中。当 `convertView` 不为 `null` 的时候，则调用 `View` 的 `getTag()`方法，把 `ViewHolder` 重新取出。这样所有控件的实例都缓存在了 `ViewHolder` 里，就没有必要每次都通过 `findViewById()`方法来获取控件实例了。

通过这两步优化之后，我们 ListView 的运行效率就已经非常不错了。

3.5.4　ListView 的点击事件

话说回来，ListView 的滚动毕竟只是满足了我们视觉上的效果，可是如果 ListView 中的子项不能点击的话，这个控件就没有什么实际的用途了。因此，本小节我们就来学习一下 ListView 如何才能响应用户的点击事件。

修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits();
        FruitAdapter adapter = new FruitAdapter(MainActivity.this, R.layout.
        fruit_item, fruitList);
        ListView listView = (ListView) findViewById(R.id.list_view);
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                                    int position, long id) {
                Fruit fruit = fruitList.get(position);
                Toast.makeText(MainActivity.this, fruit.getName(),
                    Toast.LENGTH_SHORT).show();
            }
        });
    }

    ...

}
```

可以看到，我们使用 `setOnItemClickListener()`方法为 ListView 注册了一个监听器，当用户点击了 ListView 中的任何一个子项时，就会回调 `onItemClick()`方法。在这个方法中可以通过 `position` 参数判断出用户点击的是哪一个子项，然后获取到相应的水果，并通过 Toast 将水果的名字显示出来。

重新运行程序，并点击一下橘子，效果如图 3.33 所示。

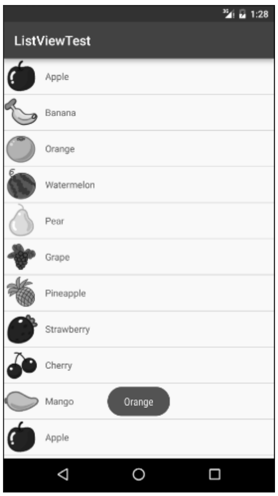


图 3.33　点击 ListView 的效果

3.6　更强大的滚动控件——RecyclerView

ListView 由于其强大的功能，在过去的 Android 开发当中可以说是贡献卓越，直到今天仍然还有不计其数的程序在继续使用着 ListView。不过 ListView 并不是完全没有缺点的，比如说如果我们不使用一些技巧来提升它的运行效率，那么 ListView 的性能就会非常差。还有，ListView 的扩展性也不够好，它只能实现数据纵向滚动的效果，如果我们想实现横向滚动的话，ListView 是做不到的。

为此，Android 提供了一个更强大的滚动控件——RecyclerView。它可以说是一个增强版的 ListView，不仅可以轻松实现和 ListView 同样的效果，还优化了 ListView 中存在的各种不足之处。目前 Android 官方更加推荐使用 RecyclerView，未来也会有更多的程序逐渐从 ListView 转向 RecyclerView，那么本节我们就来详细讲解一下 RecyclerView 的用法。

首先新建一个 RecyclerViewTest 项目，并让 Android Studio 自动帮我们创建好活动。

3.6.1　RecyclerView 的基本用法

和百分比布局类似，RecyclerView 也属于新增的控件，为了让 RecyclerView 在所有 Android 版本上都能使用，Android 团队采取了同样的方式，将 RecyclerView 定义在了 support 库当中。因此，想要使用 RecyclerView 这个控件，首先需要在项目的 build.gradle 中添加相应的依赖库才行。

打开 app/build.gradle 文件，在 `dependencies` 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:recyclerview-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

添加完之后记得要点击一下 Sync Now 来进行同步。然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

在布局中加入 RecyclerView 控件也是非常简单的，先为 RecyclerView 指定一个 id，然后将宽度和高度都设置为 `match_parent`，这样 RecyclerView 也就占满了整个布局的空间。需要注意的是，由于 RecyclerView 并不是内置在系统 SDK 当中的，所以需要把完整的包路径写出来。

这里我们想要使用 RecyclerView 来实现和 ListView 相同的效果，因此就需要准备一份同样的水果图片。简单起见，我们就直接从 ListViewTest 项目中把图片复制过来就可以了，另外顺便将 `Fruit` 类和 fruit_item.xml 也复制过来，省得将同样的代码再写一遍。

接下来需要为 RecyclerView 准备一个适配器，新建 `FruitAdapter` 类，让这个适配器继承自 `RecyclerView.Adapter`，并将泛型指定为 `FruitAdapter.ViewHolder`。其中，`ViewHolder` 是我们在 `FruitAdapter` 中定义的一个内部类，代码如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {

    private List<Fruit> mFruitList;

    static class ViewHolder extends RecyclerView.ViewHolder {
        ImageView fruitImage;
        TextView fruitName;

        public ViewHolder(View view) {
            super(view);
            fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
            fruitName = (TextView) view.findViewById(R.id.fruit_name);
        }
    }

    public FruitAdapter(List<Fruit> fruitList) {
        mFruitList = fruitList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.fruit_item, parent, false);
        ViewHolder holder = new ViewHolder(view);
        return holder;
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Fruit fruit = mFruitList.get(position);
        holder.fruitImage.setImageResource(fruit.getImageId());
        holder.fruitName.setText(fruit.getName());
    }

    @Override
    public int getItemCount() {
        return mFruitList.size();
    }

}
```

虽然这段代码看上去好像有点长，但其实它比 ListView 的适配器要更容易理解。这里我们首先定义了一个内部类 `ViewHolder`，`ViewHolder` 要继承自 `RecyclerView.ViewHolder`。然后 `ViewHolder` 的构造函数中要传入一个 `View` 参数，这个参数通常就是 RecyclerView 子项的最外

层布局，那么我们就可以通过 findViewById()方法来获取到布局中的 ImageView 和 TextView 的实例了。

接着往下看，FruitAdapter 中也有一个构造函数，这个方法用于把要展示的数据源传进来，并赋值给一个全局变量 mFruitList，我们后续的操作都将在这个数据源的基础上进行。

继续往下看，由于 FruitAdapter 是继承自 RecyclerView.Adapter 的，那么就必须重写 onCreateViewHolder()、onBindViewHolder()和 getItemCount()这 3 个方法。onCreateViewHolder()方法是用于创建 ViewHolder 实例的，我们在这个方法中将 fruit_item 布局加载进来，然后创建一个 ViewHolder 实例，并把加载出来的布局传入到构造函数当中，最后将 ViewHolder 的实例返回。onBindViewHolder()方法是用于对 RecyclerView 子项的数据进行赋值的，会在每个子项被滚动到屏幕内的时候执行，这里我们通过 position 参数得到当前项的 Fruit 实例，然后再将数据设置到 ViewHolder 的 ImageView 和 TextView 当中即可。getItemCount()方法就非常简单了，它用于告诉 RecyclerView 一共有多少子项，直接返回数据源的长度就可以了。

适配器准备好了之后，我们就可以开始使用 RecyclerView 了，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits(); // 初始化水果数据
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);
        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }

    private void initFruits() {
        for (int i = 0; i < 2; i++) {
            Fruit apple = new Fruit("Apple", R.drawable.apple_pic);
            fruitList.add(apple);
            Fruit banana = new Fruit("Banana", R.drawable.banana_pic);
            fruitList.add(banana);
            Fruit orange = new Fruit("Orange", R.drawable.orange_pic);
            fruitList.add(orange);
            Fruit watermelon = new Fruit("Watermelon", R.drawable.watermelon_pic);
            fruitList.add(watermelon);
            Fruit pear = new Fruit("Pear", R.drawable.pear_pic);
            fruitList.add(pear);
            Fruit grape = new Fruit("Grape", R.drawable.grape_pic);
            fruitList.add(grape);
            Fruit pineapple = new Fruit("Pineapple", R.drawable.pineapple_pic);
            fruitList.add(pineapple);
```

```
                Fruit strawberry = new Fruit("Strawberry", R.drawable.strawberry_pic);
                fruitList.add(strawberry);
                Fruit cherry = new Fruit("Cherry", R.drawable.cherry_pic);
                fruitList.add(cherry);
                Fruit mango = new Fruit("Mango", R.drawable.mango_pic);
                fruitList.add(mango);
            }
        }

    }
```

可以看到，这里使用了一个同样的 initFruits()方法，用于初始化所有的水果数据。接着在 onCreate()方法中我们先获取到 RecyclerView 的实例，然后创建了一个 LinearLayoutManager 对象，并将它设置到 RecyclerView 当中。LayoutManager 用于指定 RecyclerView 的布局方式，这里使用的 LinearLayoutManager 是线性布局的意思，可以实现和 ListView 类似的效果。接下来我们创建了 FruitAdapter 的实例，并将水果数据传入到 FruitAdapter 的构造函数中，最后调用 RecyclerView 的 setAdapter()方法来完成适配器设置，这样 RecyclerView 和数据之间的关联就建立完成了。

现在可以运行一下程序了，效果如图 3.34 所示。



图 3.34 RecyclerView 运行效果

可以看到，我们使用 RecyclerView 实现了和 ListView 几乎一模一样的效果，虽说在代码量方面并没有明显地减少，但是逻辑变得更加清晰了。当然这只是 RecyclerView 的基本用法而已，接下来我们就看一看 RecyclerView 还能实现哪些 ListView 实现不了的效果。

3.6.2 实现横向滚动和瀑布流布局

我们已经知道，ListView 的扩展性并不好，它只能实现纵向滚动的效果，如果想进行横向滚

动的话，ListView 就做不到了。那么 RecyclerView 就能做得到吗？当然可以，不仅能做得到，还非常简单，那么接下来我们就尝试实现一下横向滚动的效果。

首先要对 fruit_item 布局进行修改，因为目前这个布局里面的元素是水平排列的，适用于纵向滚动的场景，而如果我们要实现横向滚动的话，应该把 fruit_item 里的元素改成垂直排列才比较合理。修改 fruit_item.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="100dp"
    android:layout_height="wrap_content" >

    <ImageView
        android:id="@+id/fruit_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" />

    <TextView
        android:id="@+id/fruit_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="10dp" />

</LinearLayout>
```

可以看到，我们将 LinearLayout 改成垂直方向排列，并把宽度设为 100dp。这里将宽度指定为固定值是因为每种水果的文字长度不一致，如果用 wrap_content 的话，RecyclerView 的子项就会有长有短，非常不美观；而如果用 match_parent 的话，就会导致宽度过长，一个子项占满整个屏幕。

然后我们将 ImageView 和 TextView 都设置成了在布局中水平居中，并且使用 layout_marginTop 属性让文字和图片之间保持一些距离。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits();
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        layoutManager.setOrientation(LinearLayoutManager.HORIZONTAL);
        recyclerView.setLayoutManager(layoutManager);
```

```
        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }
    ...
}
```

MainActivity 中只加入了一行代码，调用 LinearLayoutManager 的 `setOrientation()`方法来设置布局的排列方向，默认是纵向排列的，我们传入 `LinearLayoutManager.HORIZONTAL` 表示让布局横行排列，这样 RecyclerView 就可以横向滚动了。

重新运行一下程序，效果如图 3.35 所示。

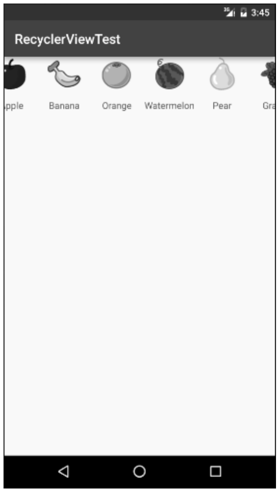


图 3.35　横向 RecyclerView 效果

你可以用手指在水平方向上滑动来查看屏幕外的数据。

为什么 ListView 很难或者根本无法实现的效果在 RecyclerView 上这么轻松就能实现了呢？这主要得益于 RecyclerView 出色的设计。ListView 的布局排列是由自身去管理的，而 RecyclerView 则将这个工作交给了 LayoutManager，LayoutManager 中制定了一套可扩展的布局排列接口，子类只要按照接口的规范来实现，就能定制出各种不同排列方式的布局了。

除了 LinearLayoutManager 之外，RecyclerView 还给我们提供了 GridLayoutManager 和 StaggeredGridLayoutManager 这两种内置的布局排列方式。GridLayoutManager 可以用于实现网格布局，StaggeredGridLayoutManager 可以用于实现瀑布流布局。这里我们来实现一下效果更加炫酷的瀑布流布局，网格布局就作为课后习题，交给你自己来研究了。

首先还是来修改一下 fruit_item.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
```

```
        android:layout_height=" wrap_content"
        android:layout_margin="5dp" >

        <ImageView
            android:id="@+id/fruit_image"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal" />

        <TextView
            android:id="@+id/fruit_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="left"
            android:layout_marginTop="10dp" />

    </LinearLayout>
```

这里做了几处小的调整，首先将 LinearLayout 的宽度由 100dp 改成了 `match_parent`，因为瀑布流布局的宽度应该是根据布局的列数来自动适配的，而不是一个固定值。另外我们使用了 `layout_margin` 属性来让子项之间互留一点间距，这样就不至于所有子项都紧贴在一些。还有就是将 TextView 的对齐属性改成了居左对齐，因为待会我们会将文字的长度变长，如果还是居中显示就会感觉怪怪的。

接着修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits();
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        StaggeredGridLayoutManager layoutManager = new
        StaggeredGridLayoutManager(3, StaggeredGridLayoutManager.VERTICAL);
        recyclerView.setLayoutManager(layoutManager);
        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }

    private void initFruits() {
        for (int i = 0; i < 2; i++) {
            Fruit apple = new Fruit(
                getRandomLengthName("Apple"), R.drawable.apple_pic);
            fruitList.add(apple);
            Fruit banana = new Fruit(
                getRandomLengthName("Banana"), R.drawable.banana_pic);
            fruitList.add(banana);
            Fruit orange = new Fruit(
```

```
                getRandomLengthName("Orange"), R.drawable.orange_pic);
            fruitList.add(orange);
            Fruit watermelon = new Fruit(
                getRandomLengthName("Watermelon"), R.drawable.watermelon_pic);
            fruitList.add(watermelon);
            Fruit pear = new Fruit(
                getRandomLengthName("Pear"), R.drawable.pear_pic);
            fruitList.add(pear);
            Fruit grape = new Fruit(
                getRandomLengthName("Grape"), R.drawable.grape_pic);
            fruitList.add(grape);
            Fruit pineapple = new Fruit(
                getRandomLengthName("Pineapple"), R.drawable.pineapple_pic);
            fruitList.add(pineapple);
            Fruit strawberry = new Fruit(
                getRandomLengthName("Strawberry"), R.drawable.strawberry_pic);
            fruitList.add(strawberry);
            Fruit cherry = new Fruit(
                getRandomLengthName("Cherry"), R.drawable.cherry_pic);
            fruitList.add(cherry);
            Fruit mango = new Fruit(
                getRandomLengthName("Mango"), R.drawable.mango_pic);
            fruitList.add(mango);
        }
    }

    private String getRandomLengthName(String name) {
        Random random = new Random();
        int length = random.nextInt(20) + 1;
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < length; i++) {
            builder.append(name);
        }
        return builder.toString();
    }

}
```

首先，在 onCreate()方法中，我们创建了一个 StaggeredGridLayoutManager 的实例。StaggeredGridLayoutManager 的构造函数接收两个参数，第一个参数用于指定布局的列数，传入 3 表示会把布局分为 3 列；第二个参数用于指定布局的排列方向，传入 StaggeredGridLayoutManager.VERTICAL 表示会让布局纵向排列，最后再把创建好的实例设置到 RecyclerView 当中就可以了，就是这么简单！

没错，仅仅修改了一行代码，我们就已经成功实现瀑布流布局的效果了。不过由于瀑布流布局需要各个子项的高度不一致才能看出明显的效果，为此我又使用了一个小技巧。这里我们把眼光聚焦在 getRandomLengthName()这个方法上，这个方法使用了 Random 对象来创造一个 1 到 20 之间的随机数，然后将参数中传入的字符串随机重复几遍。在 initFruits()方法中，每个水果的名字都改成调用 getRandomLengthName()这个方法来生成，这样就能保证各水果名字的长短

差距都比较大，子项的高度也就各不相同了。

现在重新运行一下程序，效果如图 3.36 所示。

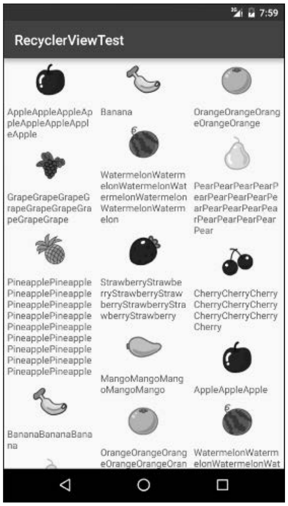


图 3.36 瀑布流布局效果

当然由于水果名字的长度每次都是随机生成的，你运行时的效果肯定和图中还是不一样的。

3.6.3 RecyclerView 的点击事件

和 ListView 一样，RecyclerView 也必须要能响应点击事件才可以，不然的话就没什么实际用途了。不过不同于 ListView 的是，RecyclerView 并没有提供类似于 `setOnItemClickListener()` 这样的注册监听器方法，而是需要我们自己给子项具体的 View 去注册点击事件，相比于 ListView 来说，实现起来要复杂一些。

那么你可能就有疑问了，为什么 RecyclerView 在各方面的设计都要优于 ListView，偏偏在点击事件上却没有处理得非常好呢？其实不是这样的，ListView 在点击事件上的处理并不人性化，`setOnItemClickListener()`方法注册的是子项的点击事件，但如果我想点击的是子项里具体的某一个按钮呢？虽然 ListView 也是能做到的，但是实现起来就相对比较麻烦了。为此，RecyclerView 干脆直接摒弃了子项点击事件的监听器，所有的点击事件都由具体的 View 去注册，就再没有这个困扰了。

下面我们来具体学习一下如何在 RecyclerView 中注册点击事件，修改 `FruitAdapter` 中的代码，如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {

    private List<Fruit> mFruitList;
```

```
    static class ViewHolder extends RecyclerView.ViewHolder {
        View fruitView;
        ImageView fruitImage;
        TextView fruitName;

        public ViewHolder(View view) {
            super(view);
            fruitView = view;
            fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
            fruitName = (TextView) view.findViewById(R.id.fruit_name);
        }
    }

    public FruitAdapter(List<Fruit> fruitList) {
        mFruitList = fruitList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.
            fruit_item, parent, false);
        final ViewHolder holder = new ViewHolder(view);
        holder.fruitView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int position = holder.getAdapterPosition();
                Fruit fruit = mFruitList.get(position);
                Toast.makeText(v.getContext(), "you clicked view " + fruit.getName(),
                    Toast.LENGTH_SHORT).show();
            }
        });
        holder.fruitImage.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int position = holder.getAdapterPosition();
                Fruit fruit = mFruitList.get(position);
                Toast.makeText(v.getContext(), "you clicked image " + fruit.getName(),
                    Toast.LENGTH_SHORT).show();
            }
        });
        return holder;
    }

    ...

}
```

我们先是修改了 `ViewHolder`，在 `ViewHolder` 中添加了 `fruitView` 变量来保存子项最外层布局的实例，然后在 `onCreateViewHolder()`方法中注册点击事件就可以了。这里分别为最外层布局和 ImageView 都注册了点击事件，RecyclerView 的强大之处也在这里，它可以轻松实现子项中任意控件或布局的点击事件。我们在两个点击事件中先获取了用户点击的 position，然后通过 position 拿到相应的 `Fruit` 实例，再使用 Toast 分别弹出两种不同的内容以示区别。

现在重新运行代码，并点击香蕉的图片部分，效果如图 3.37 所示。可以看到，这时触发了 ImageView 的点击事件。

然后再点击菠萝的文字部分，由于 TextView 并没有注册点击事件，因此点击文字这个事件会被子项的最外层布局捕获到，效果如图 3.38 所示。

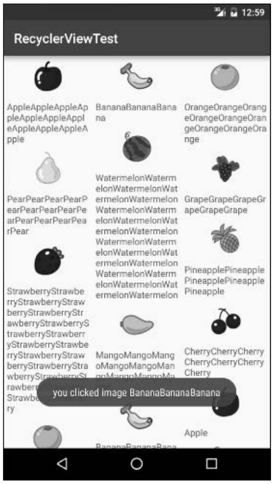


图 3.37　点击香蕉的图片部分

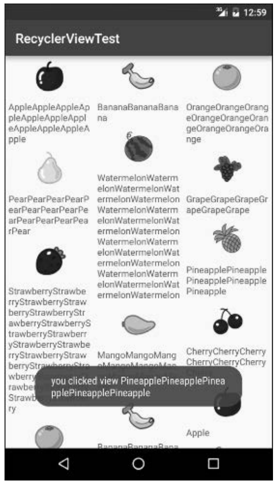


图 3.38　点击菠萝的文字部分

3.7　编写界面的最佳实践

既然已经学习了那么多 UI 开发的知识，也是时候实战一下了。这次我们要综合运用前面所学的大量内容来编写出一个较为复杂且相当美观的聊天界面，你准备好了吗？要先创建一个 UIBestPractice 项目才算准备好了哦。

3.7.1　制作 Nine-Patch 图片

在实战正式开始之前，我们还需要先学习一下如何制作 Nine-Patch 图片。你可能之前还没有听说过这个名词，它是一种被特殊处理过的 png 图片，能够指定哪些区域可以被拉伸、哪些区域不可以。

那么 Nine-Patch 图片到底有什么实际作用呢？我们还是通过一个例子来看一下吧。比如说项目中有一张气泡样式的图片 message_left.png，如图 3.39 所示。



图 3.39　气泡样式图片

我们将这张图片设置为 LinearLayout 的背景图片，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/message_left"
    >
</LinearLayout>
```

将 LinearLayout 的宽度指定为 `match_parent`，然后将它的背景图设置为 `message_left`，现在运行程序，效果如图 3.40 所示。

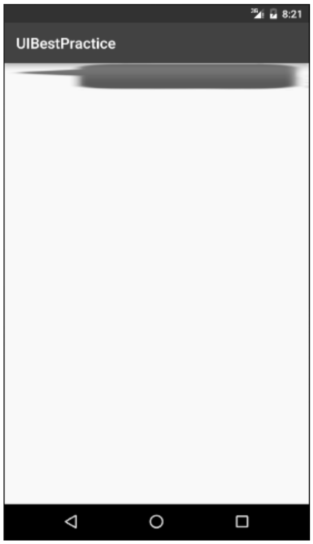


图 3.40　气泡被均匀拉伸的效果

可以看到，由于 `message_left` 的宽度不足以填满整个屏幕的宽度，整张图片被均匀地拉伸了！这种效果非常差，用户肯定是不能容忍的，这时我们就可以使用 Nine-Patch 图片来进行改善。

在 Android sdk 目录下有一个 tools 文件夹，在这个文件夹中找到 draw9patch.bat 文件，我们就是使用它来制作 Nine-Patch 图片的。不过，要打开这个文件，必须先将 JDK 的 bin 目录配置到环境变量当中才行，比如你使用的是 Android Studio 内置的 jdk，那么要配置的路径就是<Android Studio 安装目录>/jre/bin。如果你还不知道该如何配置环境变量，可以先去参考 6.4.1 小节的内容。

双击打开 draw9patch.bat 文件，在导航栏点击 File→Open 9-patch 将 message_left.png 加载进来，如图 3.41 所示。

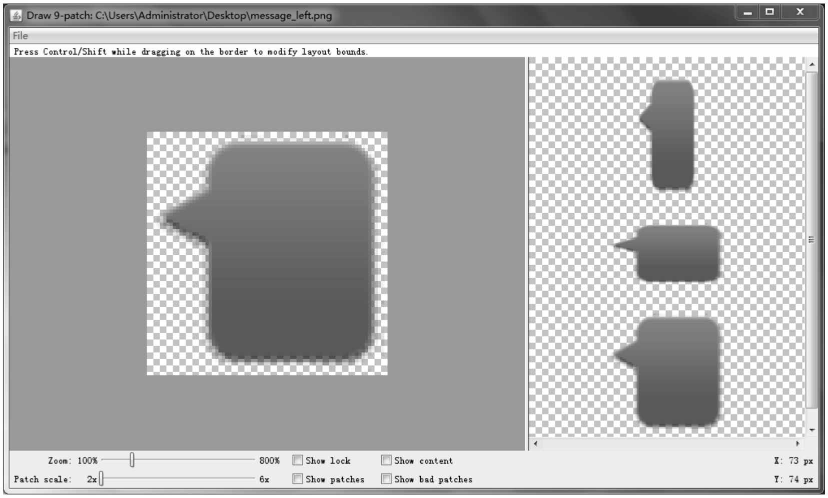


图 3.41　使用 draw9patch 编辑 message_left 图片

我们可以在图片的四个边框绘制一个个的小黑点，在上边框和左边框绘制的部分表示当图片需要拉伸时就拉伸黑点标记的区域，在下边框和右边框绘制的部分表示内容会被放置的区域。使用鼠标在图片的边缘拖动就可以进行绘制了，按住 Shift 键拖动可以进行擦除。绘制完成后效果如图 3.42 所示。

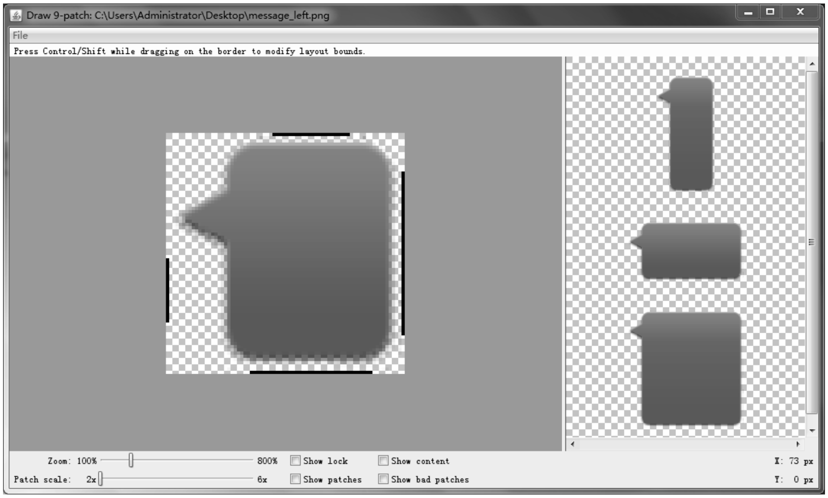


图 3.42　绘制完成后的 message_left 图片

最后点击导航栏 File→Save 9-patch 把绘制好的图片进行保存，此时的文件名就是 message_left.9.png。使用这张图片替换掉之前的 message_left.png 图片，重新运行程序，效果如图 3.43 所示。

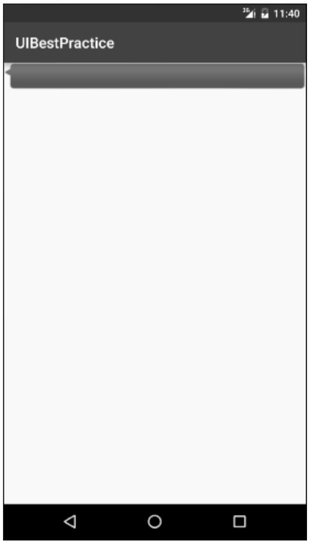


图 3.43 气泡只拉伸绘制区域的效果

这样当图片需要拉伸的时候，就可以只拉伸指定的区域，程序在外观上也有了很大的改进。有了这个知识储备之后，我们就可以进入实战环节了。

3.7.2 编写精美的聊天界面

既然是要编写一个聊天界面，那就肯定要有收到的消息和发出的消息。上一节中我们制作的 message_left.9.png 可以作为收到消息的背景图，那么毫无疑问你还需要再制作一张 message_right.9.png 作为发出消息的背景图。

图片都提供好了之后就可以开始编码了。由于待会我们会用到 RecyclerView，因此首先需要在 app/build.gradle 当中添加依赖库，如下所示：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:recyclerview-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

接下来开始编写主界面，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#d8e0e8" >

    <android.support.v7.widget.RecyclerView
```

```
        android:id="@+id/msg_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <EditText
            android:id="@+id/input_text"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:hint="Type something here"
            android:maxLines="2" />

        <Button
            android:id="@+id/send"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Send" />

    </LinearLayout>

</LinearLayout>
```

我们在主界面中放置了一个 RecyclerView 用于显示聊天的消息内容，又放置了一个 EditText 用于输入消息，还放置了一个 Button 用于发送消息。这里用到的所有属性都是我们之前学过的，相信你理解起来应该不费力。

然后定义消息的实体类，新建 `Msg`，代码如下所示：

```
public class Msg {

    public static final int TYPE_RECEIVED = 0;

    public static final int TYPE_SENT = 1;

    private String content;

    private int type;

    public Msg(String content, int type) {
        this.content = content;
        this.type = type;
    }

    public String getContent() {
        return content;
    }

    public int getType() {
```

```
        return type;
    }

}
```

Msg 类中只有两个字段，content 表示消息的内容，type 表示消息的类型。其中消息类型有两个值可选，TYPE_RECEIVED 表示这是一条收到的消息，TYPE_SENT 表示这是一条发出的消息。

接着来编写 RecyclerView 子项的布局，新建 msg_item.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="10dp" >

    <LinearLayout
        android:id="@+id/left_layout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:background="@drawable/message_left" >

        <TextView
            android:id="@+id/left_msg"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="10dp"
            android:textColor="#fff" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/right_layout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:background="@drawable/message_right" >

        <TextView
            android:id="@+id/right_msg"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="10dp" />

    </LinearLayout>

</LinearLayout>
```

这里我们让收到的消息居左对齐，发出的消息居右对齐，并且分别使用 message_left.9.png 和

message_right.9.png 作为背景图。你可能会有些疑虑，怎么能让收到的消息和发出的消息都放在同一个布局里呢？不用担心，还记得我们前面学过的可见属性吗？只要稍后在代码中根据消息的类型来决定隐藏和显示哪种消息就可以了。

接下来需要创建 RecyclerView 的适配器类，新建类 `MsgAdapter`，代码如下所示：

```
public class MsgAdapter extends RecyclerView.Adapter<MsgAdapter.ViewHolder> {

    private List<Msg> mMsgList;

    static class ViewHolder extends RecyclerView.ViewHolder {

        LinearLayout leftLayout;

        LinearLayout rightLayout;

        TextView leftMsg;

        TextView rightMsg;

        public ViewHolder(View view) {
            super(view);
            leftLayout = (LinearLayout) view.findViewById(R.id.left_layout);
            rightLayout = (LinearLayout) view.findViewById(R.id.right_layout);
            leftMsg = (TextView) view.findViewById(R.id.left_msg);
            rightMsg = (TextView) view.findViewById(R.id.right_msg);
        }
    }

    public MsgAdapter(List<Msg> msgList) {
        mMsgList = msgList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate
            (R.layout.msg_item, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Msg msg = mMsgList.get(position);
        if (msg.getType() == Msg.TYPE_RECEIVED) {
            // 如果是收到的消息，则显示左边的消息布局，将右边的消息布局隐藏
            holder.leftLayout.setVisibility(View.VISIBLE);
            holder.rightLayout.setVisibility(View.GONE);
            holder.leftMsg.setText(msg.getContent());
        } else if(msg.getType() == Msg.TYPE_SENT) {
            // 如果是发出的消息，则显示右边的消息布局，将左边的消息布局隐藏
            holder.rightLayout.setVisibility(View.VISIBLE);
            holder.leftLayout.setVisibility(View.GONE);
            holder.rightMsg.setText(msg.getContent());
```

```
        }
    }

    @Override
    public int getItemCount() {
        return mMsgList.size();
    }

}
```

以上代码你应该非常熟悉了，和我们学习 RecyclerView 那一节的代码基本是一样的，只不过在 onBindViewHolder()方法中增加了对消息类型的判断。如果这条消息是收到的，则显示左边的消息布局，如果这条消息是发出的，则显示右边的消息布局。

最后修改 MainActivity 中的代码，来为 RecyclerView 初始化一些数据，并给发送按钮加入事件响应，代码如下所示：

```
public class MainActivity extends AppCompatActivity {

    private List<Msg> msgList = new ArrayList<>();

    private EditText inputText;

    private Button send;

    private RecyclerView msgRecyclerView;

    private MsgAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initMsgs(); // 初始化消息数据
        inputText = (EditText) findViewById(R.id.input_text);
        send = (Button) findViewById(R.id.send);
        msgRecyclerView = (RecyclerView) findViewById(R.id.msg_recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        msgRecyclerView.setLayoutManager(layoutManager);
        adapter = new MsgAdapter(msgList);
        msgRecyclerView.setAdapter(adapter);
        send.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String content = inputText.getText().toString();
                if (!"".equals(content)) {
                    Msg msg = new Msg(content, Msg.TYPE_SENT);
                    msgList.add(msg);
                    adapter.notifyItemInserted(msgList.size() - 1); // 当有新消息时，
                    刷新 RecyclerView 中的显示
                    msgRecyclerView.scrollToPosition(msgList.size() - 1); // 将
                    RecyclerView 定位到最后一行
                    inputText.setText(""); // 清空输入框中的内容
```

```
                }
            }
        });
    }

    private void initMsgs() {
        Msg msg1 = new Msg("Hello guy.", Msg.TYPE_RECEIVED);
        msgList.add(msg1);
        Msg msg2 = new Msg("Hello. Who is that?", Msg.TYPE_SENT);
        msgList.add(msg2);
        Msg msg3 = new Msg("This is Tom. Nice talking to you. ", Msg.TYPE_RECEIVED);
        msgList.add(msg3);
    }

}
```

在 `initMsgs()`方法中我们先初始化了几条数据用于在 RecyclerView 中显示。然后在发送按钮的点击事件里获取了 EditText 中的内容，如果内容不为空字符串则创建出一个新的 `Msg` 对象，并把它添加到 msgList 列表中去。之后又调用了适配器的 `notifyItemInserted()`方法，用于通知列表有新的数据插入，这样新增的一条消息才能够在 RecyclerView 中显示。接着调用 RecyclerView 的 `scrollToPosition()`方法将显示的数据定位到最后一行，以保证一定可以看得到最后发出的一条消息。最后调用 EditText 的 `setText()`方法将输入的内容清空。

这样所有的工作就都完成了，终于可以检验一下我们的成果了，运行程序之后你将会看到非常美观的聊天界面，并且可以输入和发送消息，如图 3.44 所示。



图 3.44　精美的聊天界面

相信这个例子的实战过程不仅加深了你对本章中所学 UI 知识的理解，还让你有了如何灵活运用这些知识来设计出优秀界面的思路。这一章也是学了不少东西，让我们来总结一下吧。

3.8 小结与点评

虽然本章的内容很多，但我觉得学习起来应该还是挺愉快的吧。不同于上一章中我们来来回回使用那几个按钮，本章可以说是使用了各种各样的控件，制作出了丰富多彩的界面。尤其是在实战环节，编写出了那么精美的聊天界面，你的满足感应该比上一章还要强吧？

本章从 Android 中的一些常见控件开始入手，依次介绍了基本布局的用法、自定义控件的方法、ListView 的详细用法以及 RecyclerView 的使用，基本已经将重要的 UI 知识点全部覆盖了。想想在开始的时候我说不推荐使用可视化的编辑工具，而是应该全部使用 XML 的方式来编写界面，现在你是不是已经感觉使用 XML 非常简单了呢？以后不管面对多么复杂的界面，我希望你都能够自信满满，因为真正理解了界面编写的原理之后，是没有什么能够难得倒你的。

不过到目前为止，我们还只是学习了 Android 手机方面的开发技巧，下一章将会涉及一些 Android 平板方面的知识点，能够同时兼容手机和平板也是自 Android 4.0 系统开始就支持的特性。适当地放松和休息一段时间后，我们再来继续前行吧！

第 4 章

手机平板要兼顾——探究碎片

当今是移动设备发展非常迅速的时代，不仅手机已经成为了生活必需品，就连平板电脑也变得越来越普及。平板电脑和手机最大的区别就在于屏幕的大小，一般手机屏幕的大小会在 3 英寸到 6 英寸之间，而一般平板电脑屏幕的大小会在 7 英寸到 10 英寸之间。屏幕大小差距过大有可能会让同样的界面在视觉效果上有较大的差异，比如一些界面在手机上看起来非常美观，但在平板电脑上看起来就可能会有控件被过分拉长、元素之间空隙过大等情况。

作为一名专业的 Android 开发人员，能够同时兼顾手机和平板的开发是我们必须做到的事情。Android 自 3.0 版本开始引入了碎片的概念，它可以让界面在平板上更好地展示，下面我们就来一起学习一下。

4.1 碎片是什么

碎片（Fragment）是一种可以嵌入在活动当中的 UI 片段，它能让程序更加合理和充分地利用大屏幕的空间，因而在平板上应用得非常广泛。虽然碎片对你来说应该是个全新的概念，但我相信你学习起来应该毫不费力，因为它和活动实在是太像了，同样都能包含布局，同样都有自己的生命周期。你甚至可以将碎片理解成一个迷你型的活动，虽然这个迷你型的活动有可能和普通的活动是一样大的。

那么究竟要如何使用碎片才能充分地利用平板屏幕的空间呢？想象我们正在开发一个新闻应用，其中一个界面使用 RecyclerView 展示了一组新闻的标题，当点击了其中一个标题时，就打开另一个界面显示新闻的详细内容。如果是在手机中设计，我们可以将新闻标题列表放在一个活动中，将新闻的详细内容放在另一个活动中，如图 4.1 所示。

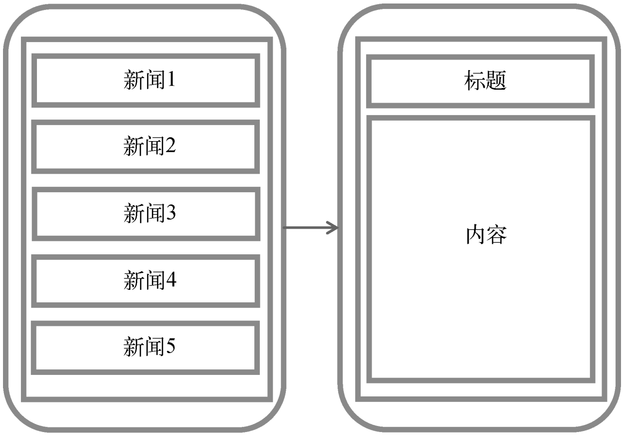


图 4.1 手机的设计方案

可是如果在平板上也这么设计，那么新闻标题列表将会被拉长至填充满整个平板的屏幕，而新闻的标题一般都不会太长，这样将会导致界面上有大量的空白区域，如图 4.2 所示。

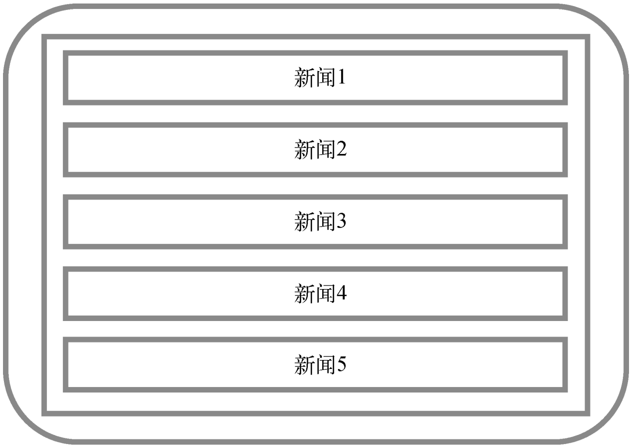


图 4.2 平板的新闻列表

因此，更好的设计方案是将新闻标题列表界面和新闻详细内容界面分别放在两个碎片中，然后在同一个活动里引入这两个碎片，这样就可以将屏幕空间充分地利用起来了，如图 4.3 所示。

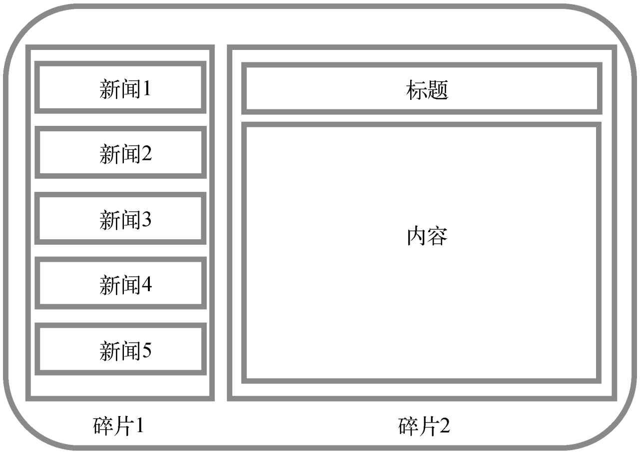


图 4.3　平板的双页设计

4.2　碎片的使用方式

介绍了这么多抽象的东西，也是时候学习一下碎片的具体用法了。你已经知道，碎片通常都是在平板开发中使用的，因此我们首先要做的就是创建一个平板模拟器。创建模拟器的方法我们在第 1 章已经学过了，创建完成后启动平板模拟器，效果如图 4.4 所示。

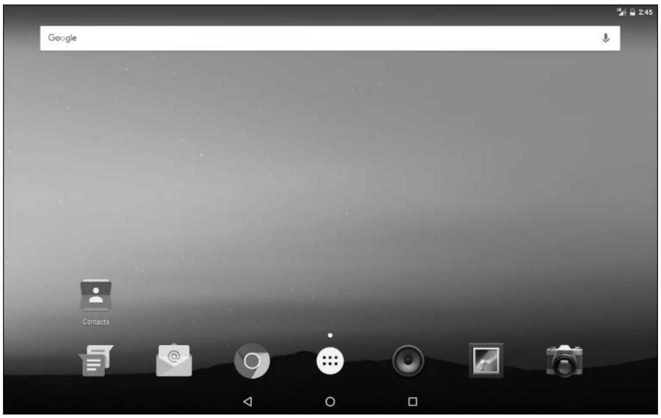


图 4.4　平板模拟器的运行效果

好了，准备工作都完成了，接着新建一个 FragmentTest 项目，然后开始我们的碎片探索之旅吧。

4.2.1　碎片的简单用法

这里我们准备先写一个最简单的碎片示例来练练手，在一个活动当中添加两个碎片，并让这

两个碎片平分活动空间。

新建一个左侧碎片布局 left_fragment.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Button"
        />

</LinearLayout>
```

这个布局非常简单，只放置了一个按钮，并让它水平居中显示。然后新建右侧碎片布局 right_fragment.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#00ff00"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="20sp"
        android:text="This is right fragment"
        />

</LinearLayout>
```

可以看到，我们将这个布局的背景色设置成了绿色，并放置了一个 TextView 用于显示一段文本。

接着新建一个 `LeftFragment` 类，并让它继承自 Fragment。注意，这里可能会有两个不同包下的 Fragment 供你选择，一个是系统内置的 android.app.Fragment，一个是 support-v4 库中的 android.support.v4.app.Fragment。这里我强烈建议你使用 support-v4 库中的 Fragment，因为它可以让碎片在所有 Android 系统版本中保持功能一致性。比如说在 Fragment 中嵌套使用 Fragment，这个功能是在 Android 4.2 系统中才开始支持的，如果你使用的是系统内置的 Fragment，那么很遗憾，4.2 系统之前的设备运行你的程序就会崩溃。而使用 support-v4 库中的 Fragment 就不会出现这个问题，只要你保证使用的是最新的 support-v4 库就可以了。另外，我们并不需要在 build.gradle 文件中添加 support-v4 库的依赖，因为 build.gradle 文件中已经添加了 appcompat-v7

库的依赖，而这个库会将 support-v4 库也一起引入进来。

现在编写一下 `LeftFragment` 中的代码，如下所示：

```
public class LeftFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.left_fragment, container, false);
        return view;
    }

}
```

这里仅仅是重写了 Fragment 的 `onCreateView()`方法，然后在这个方法中通过 LayoutInflater 的 `inflate()`方法将刚才定义的 left_fragment 布局动态加载进来，整个方法简单明了。接着我们用同样的方法再新建一个 `RightFragment`，代码如下所示：

```
public class RightFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.right_fragment, container, false);
        return view;
    }

}
```

基本上代码都是相同的，相信已经没有必要再做什么解释了。接下来修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

</LinearLayout>
```

可以看到，我们使用了<fragment>标签在布局中添加碎片，其中指定的大多数属性都是你熟悉的，只不过这里还需要通过 android:name 属性来显式指明要添加的碎片类名，注意一定要将类的包名也加上。

这样最简单的碎片示例就已经写好了，现在运行一下程序，效果如图 4.5 所示。

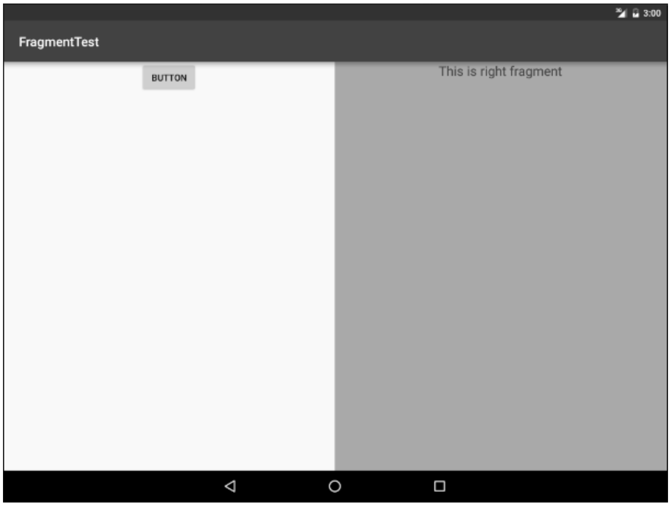


图 4.5　碎片的简单运行效果

正如我们所期待的一样，两个碎片平分了整个活动的布局。不过这个例子实在是太简单了，在真正的项目中很难有什么实际的作用，因此我们马上来看一看，关于碎片更加高级的使用技巧。

4.2.2　动态添加碎片

在上一节当中，你已经学会了在布局文件中添加碎片的方法，不过碎片真正的强大之处在于，它可以在程序运行时动态地添加到活动当中。根据具体情况来动态地添加碎片，你就可以将程序界面定制得更加多样化。

我们还是在上一节代码的基础上继续完善，新建 another_right_fragment.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#ffff00"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="20sp"
```

```
        android:text="This is another right fragment"
        />

</LinearLayout>
```

这个布局文件的代码和 right_fragment.xml 中的代码基本相同，只是将背景色改成了黄色，并将显示的文字改了改。然后新建 `AnotherRightFragment` 作为另一个右侧碎片，代码如下所示：

```
public class AnotherRightFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.another_right_fragment, container,
            false);
        return view;
    }

}
```

代码同样非常简单，在 `onCreateView()` 方法中加载了刚刚创建的 another_right_fragment 布局。这样我们就准备好了另一个碎片，接下来看一下如何将它动态地添加到活动当中。修改 activity_main.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/right_layout"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" >
    </FrameLayout>

</LinearLayout>
```

可以看到，现在将右侧碎片替换成了一个 FrameLayout 中，还记得这个布局吗？在上一章中我们学过，这是 Android 中最简单的一种布局，所有的控件默认都会摆放在布局的左上角。由于这里仅需要在布局里放入一个碎片，不需要任何定位，因此非常适合使用 FrameLayout。

下面我们将在代码中向 FrameLayout 里添加内容，从而实现动态添加碎片的功能。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this);
        replaceFragment(new RightFragment());
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                replaceFragment(new AnotherRightFragment());
                break;
            default:
                break;
        }
    }

    private void replaceFragment(Fragment fragment) {
        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction transaction = fragmentManager.beginTransaction();
        transaction.replace(R.id.right_layout, fragment);
        transaction.commit();
    }
}
```

可以看到，首先我们给左侧碎片中的按钮注册了一个点击事件，然后调用replaceFragment()方法动态添加了 RightFragment 这个碎片。当点击左侧碎片中的按钮时，又会调用 replaceFragment()方法将右侧碎片替换成 AnotherRightFragment。结合replaceFragment()方法中的代码可以看出，动态添加碎片主要分为 5 步。

(1) 创建待添加的碎片实例。

(2) 获取 FragmentManager，在活动中可以直接通过调用 getSupportFragmentManager()方法得到。

(3) 开启一个事务，通过调用 beginTransaction()方法开启。

(4) 向容器内添加或替换碎片，一般使用 replace()方法实现，需要传入容器的 id 和待添加的碎片实例。

(5) 提交事务，调用 commit()方法来完成。

这样就完成了在活动中动态添加碎片的功能，重新运行程序，可以看到和之前相同的界面，然后点击一下按钮，效果如图 4.6 所示。

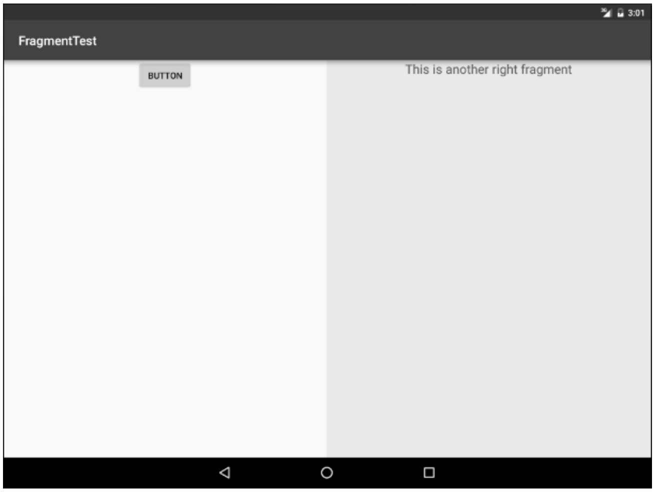


图 4.6　动态添加碎片的效果

4.2.3　在碎片中模拟返回栈

在上一小节中，我们成功实现了向活动中动态添加碎片的功能，不过你尝试一下就会发现，通过点击按钮添加了一个碎片之后，这时按下 Back 键程序就会直接退出。如果这里我们想模仿类似于返回栈的效果，按下 Back 键可以回到上一个碎片，该如何实现呢?

其实很简单，FragmentTransaction 中提供了一个 `addToBackStack()`方法，可以用于将一个事务添加到返回栈中，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    private void replaceFragment(Fragment fragment) {
        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction transaction = fragmentManager.beginTransaction();
        transaction.replace(R.id.right_layout, fragment);
        transaction.addToBackStack(null);
        transaction.commit();
    }

}
```

这里我们在事务提交之前调用了 FragmentTransaction 的 `addToBackStack()`方法，它可以接收一个名字用于描述返回栈的状态，一般传入 `null` 即可。现在重新运行程序，并点击按钮将 AnotherRightFragment 添加到活动中，然后按下 Back 键，你会发现程序并没有退出，而是回到了 RightFragment 界面，继续按下 Back 键，RightFragment 界面也会消失，再次按下 Back 键，程序才会退出。

4.2.4　碎片和活动之间进行通信

虽然碎片都是嵌入在活动中显示的，可是实际上它们的关系并没有那么亲密。你可以看出，碎片和活动都是各自存在于一个独立的类当中的，它们之间并没有那么明显的方式来直接进行通信。如果想要在活动中调用碎片里的方法，或者在碎片中调用活动里的方法，应该如何实现呢？

为了方便碎片和活动之间进行通信，FragmentManager 提供了一个类似于 findViewById() 的方法，专门用于从布局文件中获取碎片的实例，代码如下所示：

```
RightFragment rightFragment = (RightFragment) getSupportFragmentManager()
    .findFragmentById(R.id.right_fragment);
```

调用 FragmentManager 的 findFragmentById()方法，可以在活动中得到相应碎片的实例，然后就能轻松地调用碎片里的方法了。

掌握了如何在活动中调用碎片里的方法，那在碎片中又该怎样调用活动里的方法呢？其实这就更简单了，在每个碎片中都可以通过调用 getActivity()方法来得到和当前碎片相关联的活动实例，代码如下所示：

```
MainActivity activity = (MainActivity) getActivity();
```

有了活动实例之后，在碎片中调用活动里的方法就变得轻而易举了。另外当碎片中需要使用 Context 对象时，也可以使用 getActivity()方法，因为获取到的活动本身就是一个 Context 对象。

这时不知道你心中会不会产生一个疑问：既然碎片和活动之间的通信问题已经解决了，那么碎片和碎片之间可不可以进行通信呢？

说实在的，这个问题并没有看上去那么复杂，它的基本思路非常简单，首先在一个碎片中可以得到与它相关联的活动，然后再通过这个活动去获取另外一个碎片的实例，这样也就实现了不同碎片之间的通信功能，因此这里我们的答案是肯定的。

4.3　碎片的生命周期

和活动一样，碎片也有自己的生命周期，并且它和活动的生命周期实在是太像了，我相信你很快就能学会，下面我们马上就来看一下。

4.3.1　碎片的状态和回调

还记得每个活动在其生命周期内可能会有哪几种状态吗？没错，一共有运行状态、暂停状态、停止状态和销毁状态这 4 种。类似地，每个碎片在其生命周期内也可能会经历这几种状态，只不过在一些细小的地方会有部分区别。

1. 运行状态

当一个碎片是可见的，并且它所关联的活动正处于运行状态时，该碎片也处于运行状态。

2. 暂停状态

当一个活动进入暂停状态时（由于另一个未占满屏幕的活动被添加到了栈顶），与它相关联的可见碎片就会进入到暂停状态。

3. 停止状态

当一个活动进入停止状态时，与它相关联的碎片就会进入到停止状态，或者通过调用 FragmentTransaction 的 `remove()`、`replace()`方法将碎片从活动中移除，但如果在事务提交之前调用 `addToBackStack()`方法，这时的碎片也会进入到停止状态。总的来说，进入停止状态的碎片对用户来说是完全不可见的，有可能会被系统回收。

4. 销毁状态

碎片总是依附于活动而存在的，因此当活动被销毁时，与它相关联的碎片就会进入到销毁状态。或者通过调用 FragmentTransaction 的 `remove()`、`replace()`方法将碎片从活动中移除，但在事务提交之前并没有调用 `addToBackStack()`方法，这时的碎片也会进入到销毁状态。

结合之前的活动状态，相信你理解起来应该毫不费力吧。同样地，`Fragment` 类中也提供了一系列的回调方法，以覆盖碎片生命周期的每个环节。其中，活动中有的回调方法，碎片中几乎都有，不过碎片还提供了一些附加的回调方法，那我们就重点看一下这几个回调。

- ❑ `onAttach()`。当碎片和活动建立关联的时候调用。
- ❑ `onCreateView()`。为碎片创建视图（加载布局）时调用。
- ❑ `onActivityCreated()`。确保与碎片相关联的活动一定已经创建完毕的时候调用。
- ❑ `onDestroyView()`。当与碎片关联的视图被移除的时候调用。
- ❑ `onDetach()`。当碎片和活动解除关联的时候调用。

碎片完整的生命周期示意图可参考图 4.7，图片源自 Android 官网。

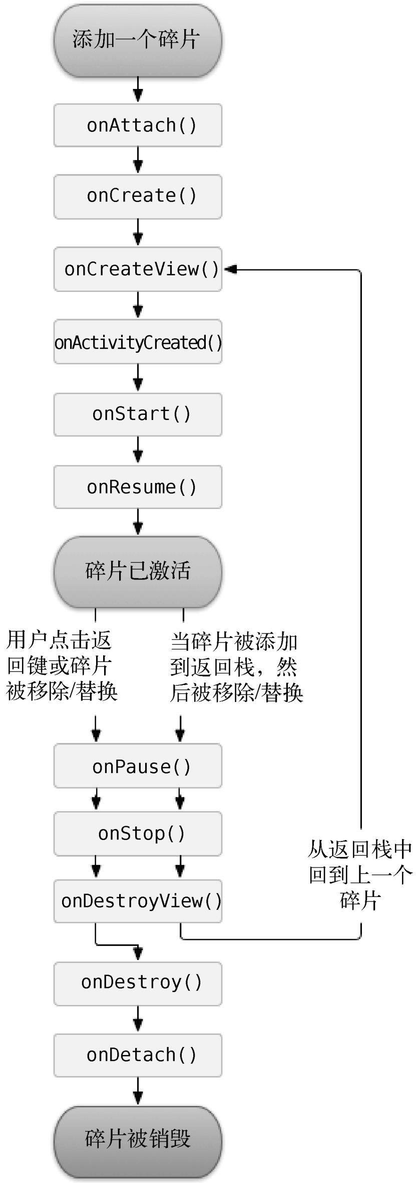


图 4.7　碎片的生命周期

4.3.2　体验碎片的生命周期

为了让你能够更加直观地体验碎片的生命周期，我们还是通过一个例子来实践一下。例子很简单，仍然是在 FragmentTest 项目的基础上改动的。

修改 RightFragment 中的代码，如下所示：

```
public class RightFragment extends Fragment {

    public static final String TAG = "RightFragment";

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        Log.d(TAG, "onAttach");
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        Log.d(TAG, "onCreateView");
        View view = inflater.inflate(R.layout.right_fragment, container, false);
        return view;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        Log.d(TAG, "onActivityCreated");
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop");
```

```
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        Log.d(TAG, "onDestroyView");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }

    @Override
    public void onDetach() {
        super.onDetach();
        Log.d(TAG, "onDetach");
    }

}
```

我们在 RightFragment 中的每一个回调方法里都加入了打印日志的代码，然后重新运行程序，这时观察 logcat 中的打印信息，如图 4.8 所示。

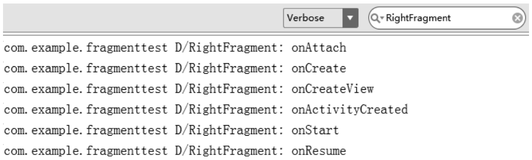


图 4.8　启动程序时的打印日志

可以看到，当 RightFragment 第一次被加载到屏幕上时，会依次执行 `onAttach()`、`onCreate()`、`onCreateView()`、`onActivityCreated()`、`onStart()`和 `onResume()`方法。然后点击 LeftFragment 中的按钮，此时打印信息如图 4.9 所示。

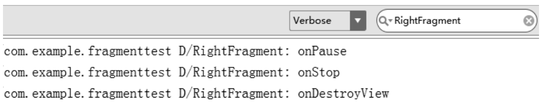


图 4.9　替换成 AnotherRightFragment 时的打印日志

由于 AnotherRightFragment 替换了 RightFragment，此时的 RightFragment 进入了停止状态，因此 `onPause()`、`onStop()`和 `onDestroyView()`方法会得到执行。当然如果在替换的时候没有调用 `addToBackStack()`方法，此时的 RightFragment 就会进入销毁状态，`onDestroy()`和 `onDetach()`方法就会得到执行。

接着按下 Back 键，RightFragment 会重新回到屏幕，打印信息如图 4.10 所示。

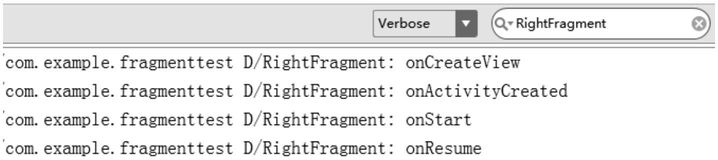


图 4.10　返回 RightFragment 时的打印日志

由于 RightFragment 重新回到了运行状态，因此 `onCreateView()`、`onActivityCreated()`、`onStart()`和 `onResume()`方法会得到执行。注意此时 `onCreate()`方法并不会执行，因为我们借助了 `addToBackStack()`方法使得 RightFragment 并没有被销毁。

现在再次按下 Back 键，打印信息如图 4.11 所示。

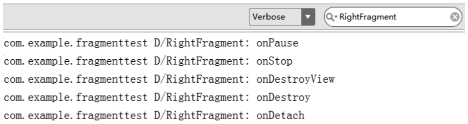


图 4.11　退出程序时的打印日志

依次会执行 `onPause()`、`onStop()`、`onDestroyView()`、`onDestroy()`和 `onDetach()`方法，最终将碎片销毁掉。这样碎片完整的生命周期你也体验了一遍，是不是理解得更加深刻了？

另外值得一提的是，在碎片中你也是可以通过 `onSaveInstanceState()`方法来保存数据的，因为进入停止状态的碎片有可能在系统内存不足的时候被回收。保存下来的数据在 `onCreate()`、`onCreateView()`和 `onActivityCreated()`这 3 个方法中你都可以重新得到，它们都含有一个 Bundle 类型的 `savedInstanceState` 参数。具体的代码我就不在这里给出了，如果你忘记了该如何编写，可以参考 2.4.5 小节。

4.4　动态加载布局的技巧

虽然动态添加碎片的功能很强大，可以解决很多实际开发中的问题，但是它毕竟只是在一个布局文件中进行一些添加和替换操作。如果程序能够根据设备的分辨率或屏幕大小在运行时来决定加载哪个布局，那我们可发挥的空间就更多了。因此本节我们就来探讨一下 Android 中动态加载布局的技巧。

4.4.1　使用限定符

如果你经常使用平板电脑，应该会发现现在很多的平板应用都采用的是双页模式（程序会在左侧的面板上显示一个包含子项的列表，在右侧的面板上显示内容），因为平板电脑的屏幕足够大，完全可以同时显示下两页的内容，但手机的屏幕一次就只能显示一页的内容，因此两个页面需要分开显示。

那么怎样才能在运行时判断程序应该是使用双页模式还是单页模式呢？这就需要借助限定符（Qualifiers）来实现了。下面我们通过一个例子来学习一下它的用法，修改 FragmentTest 项目中的 activity_main.xml 文件，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

这里将多余的代码都删掉，只留下一个左侧碎片，并让它充满整个父布局。接着在 res 目录下新建 layout-large 文件夹，在这个文件夹下新建一个布局，也叫作 activity_main.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />

</LinearLayout>
```

可以看到，layout/activity_main 布局只包含了一个碎片，即单页模式，而 layout-large/activity_main 布局包含了两个碎片，即双页模式。其中 `large` 就是一个限定符，那些屏幕被认为是 `large` 的设备就会自动加载 layout-large 文件夹下的布局，而小屏幕的设备则还是会加载 layout 文件夹下的布局。

然后将 MainActivity 中 `replaceFragment()`方法里的代码注释掉，并在平板模拟器上重新运行程序，效果如图 4.12 所示。

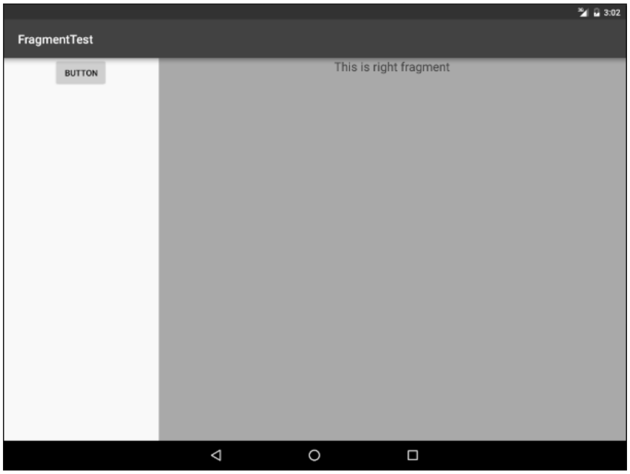


图 4.12　双页模式运行效果

再启动一个手机模拟器，并在这个模拟器上重新运行程序，效果如图 4.13 所示。

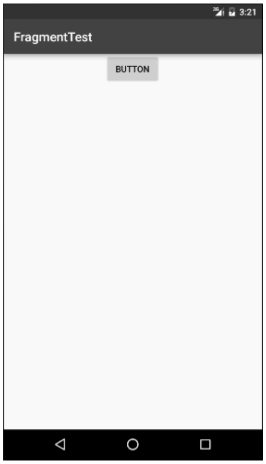


图 4.13　单页模式运行效果

这样我们就实现了在程序运行时动态加载布局的功能。

Android 中一些常见的限定符可以参考下表。

屏幕特征	限定符	描　　述
大小	`small`	提供给小屏幕设备的资源
	`normal`	提供给中等屏幕设备的资源
	`large`	提供给大屏幕设备的资源
	`xlarge`	提供给超大屏幕设备的资源

（续）

屏幕特征	限定符	描　　述
分辨率	ldpi	提供给低分辨率设备的资源（120dpi以下）
	mdpi	提供给中等分辨率设备的资源（120dpi~160dpi）
	hdpi	提供给高分辨率设备的资源（160dpi~240dpi）
	xhdpi	提供给超高分辨率设备的资源（240dpi~320dpi）
	xxhdpi	提供给超超高分辨率设备的资源（320dpi~480dpi）
方向	land	提供给横屏设备的资源
	port	提供给竖屏设备的资源

4.4.2　使用最小宽度限定符

在上一小节中我们使用 `large` 限定符成功解决了单页双页的判断问题，不过很快又有一个新的问题出现了，`large` 到底是指多大呢？有的时候我们希望可以更加灵活地为不同设备加载布局，不管它们是不是被系统认定为 `large`，这时就可以使用最小宽度限定符（Smallest-width Qualifier）了。

最小宽度限定符允许我们对屏幕的宽度指定一个最小值（以 dp 为单位），然后以这个最小值为临界点，屏幕宽度大于这个值的设备就加载一个布局，屏幕宽度小于这个值的设备就加载另一个布局。

在 res 目录下新建 layout-sw600dp 文件夹，然后在这个文件夹下新建 activity_main.xml 布局，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />

</LinearLayout>
```

这就意味着，当程序运行在屏幕宽度大于 600dp 的设备上时，会加载 layout-sw600dp/activity_main 布局，当程序运行在屏幕宽度小于 600dp 的设备上时，则仍然加载默认的 layout/activity_main 布局。

4.5　碎片的最佳实践——一个简易版的新闻应用

现在你已经将关于碎片的重要知识点都掌握得差不多了，不过在灵活运用方面可能还有些欠缺，因此下面该进入我们本章的最佳实践环节了。

前面有提到过，碎片很多时候都是在平板开发当中使用的，主要是为了解决屏幕空间不能充分利用的问题。那是不是就表明，我们开发的程序都需要提供一个手机版和一个 Pad 版呢？确实有不少公司都是这么做的，但是这样会浪费很多的人力物力。因为维护两个版本的代码成本很高，每当增加什么新功能时，需要在两份代码里各写一遍，每当发现一个 bug 时，需要在两份代码里各修改一次。因此今天我们最佳实践的内容就是，教你如何编写同时兼容手机和平板的应用程序。

还记得在本章开始的时候提到过的一个新闻应用吗？现在我们就将运用本章中所学的知识来编写一个简易版的新闻应用，并且要求它是可以同时兼容手机和平板的。新建好一个 FragmentBestPractice 项目，然后开始动手吧！

由于待会在编写新闻列表时会使用到 RecyclerView，因此首先需要在 app/build.gradle 当中添加依赖库，如下所示：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:recyclerview-v7:24.2.1'
}
```

接下来我们要准备好一个新闻的实体类，新建类 News，代码如下所示：

```
public class News {

    private String title;

    private String content;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

}
```

News 类的代码还是比较简单的，title 字段表示新闻标题，content 字段表示新闻内容。接着新建布局文件 news_content_frag.xml，用于作为新闻内容的布局：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:id="@+id/visibility_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:visibility="invisible" >

        <TextView
            android:id="@+id/news_title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:padding="10dp"
            android:textSize="20sp" />

        <View
            android:layout_width="match_parent"
            android:layout_height="1dp"
            android:background="#000" />

        <TextView
            android:id="@+id/news_content"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:padding="15dp"
            android:textSize="18sp" />

    </LinearLayout>

    <View
        android:layout_width="1dp"
        android:layout_height="match_parent"
        android:layout_alignParentLeft="true"
        android:background="#000" />

</RelativeLayout>
```

新闻内容的布局主要可以分为两个部分，头部部分显示新闻标题，正文部分显示新闻内容，中间使用一条细线分隔开。这里的细线是利用 View 来实现的，将 View 的宽或高设置为 1dp，再通过 background 属性给细线设置一下颜色就可以了。这里我们把细线设置成黑色。

然后再新建一个 NewsContentFragment 类，继承自 Fragment，代码如下所示：

```
public class NewsContentFragment extends Fragment {
```

```
    private View view;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        view = inflater.inflate(R.layout.news_content_frag, container, false);
        return view;
    }

    public void refresh(String newsTitle, String newsContent) {
        View visibilityLayout = view.findViewById(R.id.visibility_layout);
        visibilityLayout.setVisibility(View.VISIBLE);
        TextView newsTitleText = (TextView) view.findViewById (R.id.news_title);
        TextView newsContentText = (TextView) view.findViewById(R.id.news_content);
        newsTitleText.setText(newsTitle); // 刷新新闻的标题
        newsContentText.setText(newsContent); // 刷新新闻的内容
    }

}
```

首先在 `onCreateView()`方法里加载了我们刚刚创建的 news_content_frag 布局，这个没什么好解释的。接下来又提供了一个 `refresh()`方法，这个方法就是用于将新闻的标题和内容显示在界面上的。可以看到，这里通过 `findViewById()`方法分别获取到新闻标题和内容的控件，然后将方法传递进来的参数设置进去。

这样我们就把新闻内容的碎片和布局都创建好了，但是它们都是在双页模式中使用的，如果想在单页模式中使用的话，我们还需要再创建一个活动。右击 com.example.fragmentbestpractice 包→New→Activity→Empty Activity，新建一个 NewsContentActivity，并将布局名指定成 news_content，然后修改 news_content.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id="@+id/news_content_fragment"
        android:name="com.example.fragmentbestpractice.NewsContentFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</LinearLayout>
```

这里我们充分发挥了代码的复用性，直接在布局中引入了 `NewsContentFragment`，这样也就相当于把 news_content_frag 布局的内容自动加了进来。

然后修改 `NewsContentActivity` 中的代码，如下所示：

```
public class NewsContentActivity extends AppCompatActivity {
```

```
    public static void actionStart(Context context, String newsTitle, String
        newsContent) {
        Intent intent = new Intent(context, NewsContentActivity.class);
        intent.putExtra("news_title", newsTitle);
        intent.putExtra("news_content", newsContent);
        context.startActivity(intent);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_content);
        String newsTitle = getIntent().getStringExtra("news_title"); // 获取传入的新
        闻标题
        String newsContent = getIntent().getStringExtra("news_content"); // 获取传入
        的新闻内容
        NewsContentFragment newsContentFragment = (NewsContentFragment)
        getSupportFragmentManager().findFragmentById(R.id.news_content_fragment);
        newsContentFragment.refresh(newsTitle, newsContent); //刷新 NewsContent-
            Fragment 界面
    }

}
```

可以看到，在 `onCreate()`方法中我们通过 Intent 获取到了传入的新闻标题和新闻内容，然后调用 `FragmentManager` 的 `findFragmentById()`方法得到了 `NewsContentFragment` 的实例，接着调用它的 `refresh()`方法，并将新闻的标题和内容传入，就可以把这些数据显示出来了。注意这里我们还提供了一个 `actionStart()`方法，还记得它的作用吗？如果忘记的话就再去阅读一遍 2.6.3 小节吧。

接下来还需要再创建一个用于显示新闻列表的布局，新建 news_title_frag.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/news_title_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</LinearLayout>
```

这个布局的代码就非常简单了，里面只有一个用于显示新闻列表的 RecyclerView。既然要用到 RecyclerView，那么就必定少不了子项的布局。新建 news_item.xml 作为 RecyclerView 子项的布局，代码如下所示：

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:id="@+id/news_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:maxLines="true"
        android:ellipsize="end"
        android:textSize="18sp"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:paddingTop="15dp"
        android:paddingBottom="15dp" />
```

子项的布局也非常简单，只有一个 TextView。仔细观察 TextView，你会发现其中有几个属性是我们之前没有学过的。`android:padding` 表示给控件的周围加上补白，这样不至于让文本内容会紧靠在边缘上。`android:maxLines` 设置为 `true` 表示让这个 TextView 只能单行显示。`android:ellipsize` 用于设定当文本内容超出控件宽度时，文本的缩略方式，这里指定成 `end` 表示在尾部进行缩略。

既然新闻列表和子项的布局都已经创建好了，那么接下来我们就需要一个用于展示新闻列表的地方。这里新建 `NewsTitleFragment` 作为展示新闻列表的碎片，代码如下所示：

```
public class NewsTitleFragment extends Fragment {

    private boolean isTwoPane;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.news_title_frag, container, false);
        return view;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        if (getActivity().findViewById(R.id.news_content_layout) != null) {
            isTwoPane = true; // 可以找到 news_content_layout 布局时，为双页模式
        } else {
            isTwoPane = false; // 找不到 news_content_layout 布局时，为单页模式
        }
    }

}
```

可以看到，`NewsTitleFragment` 中并没有多少代码，在 `onCreateView()`方法中加载了 news_title_frag 布局，这个没什么好说的。我们注意看一下 `onActivityCreated()`方法，这个方法通过在活动中能否找到一个 id 为 news_content_layout 的 View 来判断当前是双页模式还是单页模式，因此我们需要让这个 id 为 news_content_layout 的 View 只在双页模式中才会出现。

那么怎样才能实现这个功能呢？其实并不复杂，只需要借助我们刚刚学过的限定符就可以

了。首先修改 activity_main.xml 中的代码，如下所示：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/news_title_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/news_title_fragment"
        android:name="com.example.fragmentbestpractice.NewsTitleFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</FrameLayout>
```

上述代码表示，在单页模式下，只会加载一个新闻标题的碎片。

然后新建 layout-sw600dp 文件夹，在这个文件夹下再新建一个 activity_main.xml 文件，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/news_title_fragment"
        android:name="com.example.fragmentbestpractice.NewsTitleFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/news_content_layout"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" >

        <fragment
            android:id="@+id/news_content_fragment"
            android:name="com.example.fragmentbestpractice.NewsContentFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </FrameLayout>

</LinearLayout>
```

可以看出，在双页模式下我们同时引入了两个碎片，并将新闻内容的碎片放在了一个 FrameLayout 布局下，而这个布局的 id 正是 news_content_layout。因此，能够找到这个 id 的时候就是双页模式，否则就是单面模式。

现在我们已经将绝大部分的工作都完成了，但还剩下至关重要的一点，就是在 NewsTitle-

Fragment 中通过 RecyclerView 将新闻列表展示出来。我们在 NewsTitleFragment 中新建一个内部类 NewsAdapter 来作为 RecyclerView 的适配器，如下所示：

```
public class NewsTitleFragment extends Fragment {

    private boolean isTwoPane;

    ...

    class NewsAdapter extends RecyclerView.Adapter<NewsAdapter.ViewHolder> {

        private List<News> mNewsList;

        class ViewHolder extends RecyclerView.ViewHolder {

            TextView newsTitleText;

            public ViewHolder(View view) {
                super(view);
                newsTitleText = (TextView) view.findViewById(R.id.news_title);
            }

        }

        public NewsAdapter(List<News> newsList) {
            mNewsList = newsList;
        }

        @Override
        public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.news_item, parent, false);
            final ViewHolder holder = new ViewHolder(view);
            view.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    News news = mNewsList.get(holder.getAdapterPosition());
                    if (isTwoPane) {
                    // 如果是双页模式，则刷新 NewsContentFragment 中的内容
                        NewsContentFragment newsContentFragment =
                            (NewsContentFragment)  getFragmentManager()
                            .findFragmentById(R.id.news_content_fragment);
                        newsContentFragment.refresh(news.getTitle(),
                        news.getContent());
                    } else {
                        // 如果是单页模式，则直接启动 NewsContentActivity
                        NewsContentActivity.actionStart(getActivity(),
                        news.getTitle(), news.getContent());
                    }
                }
            });
            return holder;
        }
```

```
        @Override
        public void onBindViewHolder(ViewHolder holder, int position) {
            News news = mNewsList.get(position);
            holder.newsTitleText.setText(news.getTitle());
        }

        @Override
        public int getItemCount() {
            return mNewsList.size();
        }

    }

}
```

RecyclerView 的用法你已经相当熟练了，因此这个适配器的代码对你来说应该没有什么难度吧？需要注意的是，之前我们都是将适配器写成一个独立的类，其实也是可以写成内部类的，这里写成内部类的好处就是可以直接访问 NewsTitleFragment 的变量，比如 isTwoPane。

观察一下 onCreateViewHolder()方法中注册的点击事件，首先获取到了点击项的 News 实例，然后通过 isTwoPane 变量来判断当前是单页还是双页模式，如果是单页模式，就启动一个新的活动去显示新闻内容，如果是双页模式，就更新新闻内容碎片里的数据。

现在还剩最后一步收尾工作，就是向 RecyclerView 中填充数据了。修改 NewsTitle-Fragment 中的代码，如下所示：

```
public class NewsTitleFragment extends Fragment {

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.news_title_frag, container, false);
        RecyclerView newsTitleRecyclerView = (RecyclerView) view.findViewById
            (R.id.news_title_recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(getActivity());
        newsTitleRecyclerView.setLayoutManager(layoutManager);
        NewsAdapter adapter = new NewsAdapter(getNews());
        newsTitleRecyclerView.setAdapter(adapter);
        return view;
    }

    private List<News> getNews() {
        List<News> newsList = new ArrayList<>();
        for (int i = 1; i <= 50; i++) {
            News news = new News();
            news.setTitle("This is news title " + i);
            news.setContent(getRandomLengthContent("This is news content " + i + "."));
            newsList.add(news);
        }
```

```
            return newsList;
        }

        private String getRandomLengthContent(String content) {
            Random random = new Random();
            int length = random.nextInt(20) + 1;
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < length; i++) {
                builder.append(content);
            }
            return builder.toString();
        }

        ...

    }
```

可以看到，onCreateView()方法中添加了 RecyclerView 标准的使用方法，在碎片中使用 RecyclerView 和在活动中使用几乎是一模一样的，相信没有什么需要解释的。另外，这里调用了 getNews()方法来初始化 50 条模拟新闻数据，同样使用了一个 getRandomLengthContent()方法来随机生成新闻内容的长度，以保证每条新闻的内容差距比较大，相信你对这个方法肯定不会陌生了。

这样我们所有的编写工作就已经完成了，赶快来运行一下吧！首先在手机模拟器上运行，效果如图 4.14 所示。

可以看到许多条新闻的标题，然后点击第一条新闻，会启动一个新的活动来显示新闻的内容，效果如图 4.15 所示。



图 4.14　单页模式的新闻列表界面



图 4.15　单页模式的新闻内容界面

接下来将程序在平板模拟器上运行，同样点击第一条新闻，效果如图 4.16 所示。

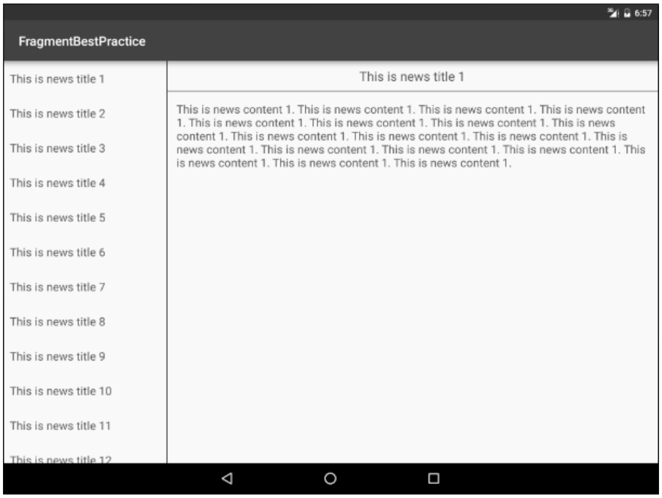


图 4.16　双页模式的新闻标题和内容界面

怎么样？同样的一份代码，在手机和平板上运行却分别是两种完全不同的效果，说明我们程序的兼容性已经相当不错了。通过这个例子，我相信你对碎片的理解一定又加深了很多，现在就让我们一起来总结一下吧。

4.6　小结与点评

你应该可以感觉到，上一节中我们开发的新闻应用，代码复杂度还是有点高的，比起只需要兼容一个终端的应用，我们要考虑的东西多了很多。不过在开发的过程中多付出一些，在以后的代码维护中就可以轻松很多。因此，有时候提前的付出还是很值得的。

我们再来回顾一下本章所学的内容吧，首先你了解了碎片的基本概念以及使用场景，接着通过几个实例掌握了碎片的常见用法，随后又学习了碎片生命周期的相关内容以及动态加载布局的技巧，最后在本章的最佳实践部分将前面所学的内容综合运用了一遍，相信你已经将碎片相关知识点都牢记在心，并可以较为熟练地应用了。

本章其实是具有一个里程碑式的纪念意义的，因为到这里为止，我们已经基本将 Android UI 相关的重要知识点都讲完了。后面在很长一段时间内都不会再系统性地介绍 UI 方面的知识，而是将结合前面所学的 UI 知识来更好地讲解相应章节的内容。那么我们下一章将要学习什么呢？还记得在第 1 章里介绍过的 Android 四大组件吧？目前我们只掌握了活动这一个组件，那么下一章就来学习广播接收器吧。跟上脚步，准备继续前进！

第 5 章

全局大喇叭——详解广播机制

记得在我上学的时候，每个班级的教室里都会装有一个喇叭，这些喇叭都是接入到学校的广播室的，一旦有什么重要的通知，就会播放一条广播来告知全校的师生。类似的工作机制其实在计算机领域也有很广泛的应用，如果你了解网络通信原理应该会知道，在一个 IP 网络范围中，最大的 IP 地址是被保留作为广播地址来使用的。比如某个网络的 IP 范围是 192.168.0.XXX，子网掩码是 255.255.255.0，那么这个网络的广播地址就是 192.168.0.255。广播数据包会被发送到同一网络上的所有端口，这样在该网络中的每台主机都将会收到这条广播。

为了便于进行系统级别的消息通知，Android 也引入了一套类似的广播消息机制。相比于我前面举出的两个例子，Android 中的广播机制会显得更加灵活，本章就将对这一机制的方方面面进行详细的讲解。

5.1 广播机制简介

为什么说 Android 中的广播机制更加灵活呢？这是因为 Android 中的每个应用程序都可以对自己感兴趣的广播进行注册，这样该程序就只会接收到自己所关心的广播内容，这些广播可能是来自于系统的，也可能是来自于其他应用程序的。Android 提供了一套完整的 API，允许应用程序自由地发送和接收广播。发送广播的方法其实之前稍微提到过，如果你记性好的话可能还会有印象，就是借助我们第 2 章学过的 Intent。而接收广播的方法则需要引入一个新的概念——广播接收器（Broadcast Receiver）。

广播接收器的具体用法将会在下一节中做介绍，这里我们先来了解一下广播的类型。Android 中的广播主要可以分为两种类型：标准广播和有序广播。

- **标准广播**（Normal broadcasts）是一种完全异步执行的广播，在广播发出之后，所有的广播接收器几乎都会在同一时刻接收到这条广播消息，因此它们之间没有任何先后顺序可言。这种广播的效率会比较高，但同时也意味着它是无法被截断的。标准广播的工作流程如图 5.1 所示。

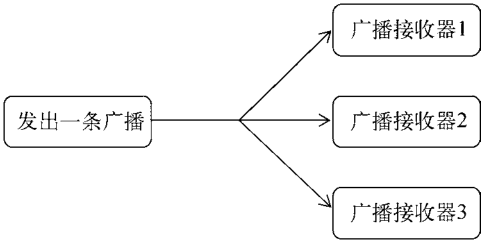


图 5.1 标准广播工作示意图

- **有序广播**（Ordered broadcasts）则是一种同步执行的广播，在广播发出之后，同一时刻只会有一个广播接收器能够收到这条广播消息，当这个广播接收器中的逻辑执行完毕后，广播才会继续传递。所以此时的广播接收器是有先后顺序的，优先级高的广播接收器就可以先收到广播消息，并且前面的广播接收器还可以截断正在传递的广播，这样后面的广播接收器就无法收到广播消息了。有序广播的工作流程如图 5.2 所示。

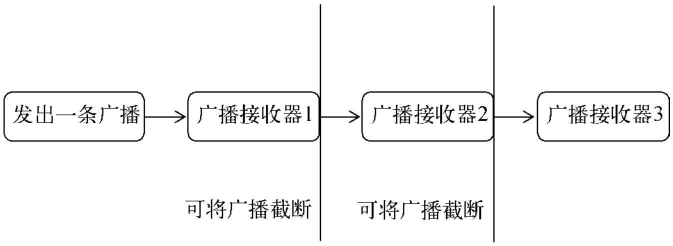


图 5.2 有序广播工作示意图

掌握了这些基本概念后，我们就可以来尝试一下广播的用法了，首先就从接收系统广播开始吧。

5.2 接收系统广播

Android 内置了很多系统级别的广播，我们可以在应用程序中通过监听这些广播来得到各种系统的状态信息。比如手机开机完成后会发出一条广播，电池的电量发生变化会发出一条广播，时间或时区发生改变也会发出一条广播，等等。如果想要接收到这些广播，就需要使用广播接收器，下面我们就来看一下它的具体用法。

5.2.1 动态注册监听网络变化

广播接收器可以自由地对自己感兴趣的广播进行注册，这样当有相应的广播发出时，广播接收器就能够收到该广播，并在内部处理相应的逻辑。注册广播的方式一般有两种，在代码中注册和在 AndroidManifest.xml 中注册，其中前者也被称为动态注册，后者也被称为静态注册。

那么该如何创建一个广播接收器呢？其实只需要新建一个类，让它继承自 `BroadcastReceiver`，并重写父类的 `onReceive()`方法就行了。这样当有广播到来时，`onReceive()`方法

就会得到执行，具体的逻辑就可以在这个方法中处理。

那我们就先通过动态注册的方式编写一个能够监听网络变化的程序，借此学习一下广播接收器的基本用法吧。新建一个 BroadcastTest 项目，然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private IntentFilter intentFilter;

    private NetworkChangeReceiver networkChangeReceiver;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        intentFilter = new IntentFilter();
        intentFilter.addAction("android.net.conn.CONNECTIVITY_CHANGE");
        networkChangeReceiver = new NetworkChangeReceiver();
        registerReceiver(networkChangeReceiver, intentFilter);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(networkChangeReceiver);
    }

    class NetworkChangeReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(context, "network changes", Toast.LENGTH_SHORT).show();
        }

    }

}
```

可以看到，我们在 MainActivity 中定义了一个内部类 NetworkChangeReceiver，这个类是继承自 BroadcastReceiver 的，并重写了父类的 onReceive()方法。这样每当网络状态发生变化时，onReceive()方法就会得到执行，这里只是简单地使用 Toast 提示了一段文本信息。

然后观察 onCreate()方法，首先我们创建了一个 IntentFilter 的实例，并给它添加了一个值为 android.net.conn.CONNECTIVITY_CHANGE 的 action，为什么要添加这个值呢？因为当网络状态发生变化时，系统发出的正是一条值为 android.net.conn.CONNECTIVITY_CHANGE 的广播，也就是说我们的广播接收器想要监听什么广播，就在这里添加相应的 action。接下来创建了一个 NetworkChangeReceiver 的实例，然后调用 registerReceiver()方法进行注册，将 NetworkChangeReceiver 的实例和 IntentFilter 的实例都传了进去，这样 NetworkChange-Receiver 就会收到所有值为 android.net.conn.CONNECTIVITY_CHANGE 的广播，也就实现了

监听网络变化的功能。

最后要记得，动态注册的广播接收器一定都要取消注册才行，这里我们是在 onDestroy() 方法中通过调用 unregisterReceiver()方法来实现的。

整体来说，代码还是非常简单的，现在运行一下程序。首先你会在注册完成的时候收到一条广播，然后按下 Home 键回到主界面（注意不能按 Back 键，否则 onDestroy()方法会执行），接着打开 Settings 程序→Data usage 进入到数据使用详情界面，然后尝试着开关 Cellular data 按钮来启动和禁用网络，你就会看到有 Toast 提醒你网络发生了变化。

不过，只是提醒网络发生了变化还不够人性化，最好是能准确地告诉用户当前是有网络还是没有网络，因此我们还需要对上面的代码进行进一步的优化。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    class NetworkChangeReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            ConnectivityManager connectionManager = (ConnectivityManager)
                getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo = connectionManager.getActiveNetworkInfo();
            if (networkInfo != null && networkInfo.isAvailable()) {
                Toast.makeText(context, "network is available",
                    Toast.LENGTH_SHORT).show();
            } else {
                Toast.makeText(context, "network is unavailable",
                    Toast.LENGTH_SHORT).show();
            }
        }

    }

}
```

在 onReceive()方法中，首先通过 getSystemService()方法得到了 ConnectivityManager 的实例，这是一个系统服务类，专门用于管理网络连接的。然后调用它的 getActiveNetwork-Info()方法可以得到 NetworkInfo 的实例，接着调用 NetworkInfo 的 isAvailable()方法，就可以判断出当前是否有网络了，最后我们还是通过 Toast 的方式对用户进行提示。

另外，这里有非常重要的一点需要说明，Android 系统为了保护用户设备的安全和隐私，做了严格的规定：如果程序需要进行一些对用户来说比较敏感的操作，就必须在配置文件中声明权限才可以，否则程序将会直接崩溃。比如这里访问系统的网络状态就是需要声明权限的。打开 AndroidManifest.xml 文件，在里面加入如下权限就可以访问系统网络状态了：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    package="com.example.broadcasttest">

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    ...
</manifest>
```

这是你第一次遇到权限的问题,其实 Android 中有许多操作都是需要声明权限才可以进行的,后面我们还会不断使用新的权限。不过目前这个访问系统网络状态的权限还是比较简单的，只需要在 AndroidManifest.xml 文件中声明一下就可以了，而 Android 6.0 系统中引入了更加严格的运行时权限,从而能够更好地保证用户设备的安全和隐私,关于这部分内容我们将在第 7 章中学习。

现在重新运行程序，然后按下 Home 键→Settings→Data usage，进入到数据使用详情界面，关闭 Cellular data 会弹出无网络可用的提示，如图 5.3 所示。

然后重新打开 Cellular data 又会弹出网络可用的提示，如图 5.4 所示。

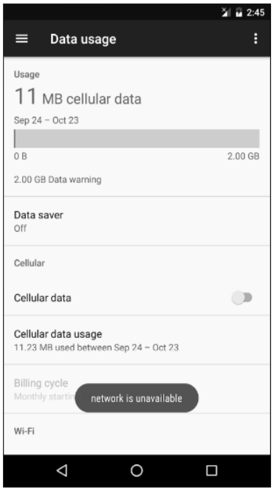


图 5.3　禁用系统网络

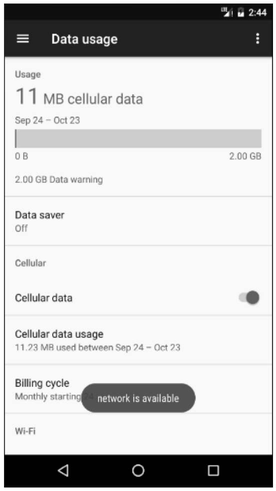


图 5.4　启用系统网络

5.2.2　静态注册实现开机启动

动态注册的广播接收器可以自由地控制注册与注销，在灵活性方面有很大的优势，但是它也存在着一个缺点,即必须要在程序启动之后才能接收到广播,因为注册的逻辑是写在 `onCreate()` 方法中的。那么有没有什么办法可以让程序在未启动的情况下就能接收到广播呢？这就需要使用静态注册的方式了。

这里我们准备让程序接收一条开机广播，当收到这条广播时就可以在 `onReceive()`方法里执行相应的逻辑，从而实现开机启动的功能。可以使用 Android Studio 提供的快捷方式来创建一个广播接收器，右击 com.example.broadcasttest 包→New→Other→Broadcast Receiver，会弹出如

图 5.5 所示的窗口。

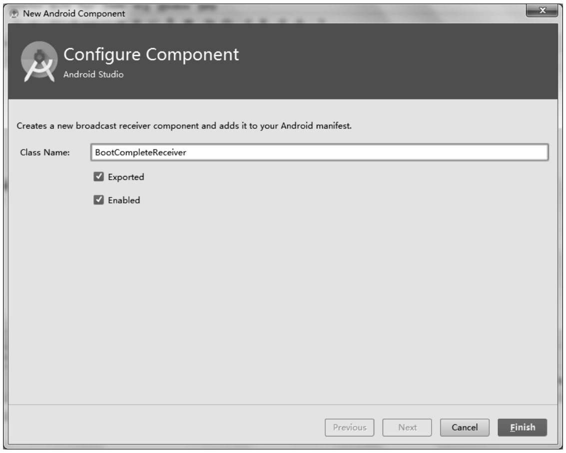


图 5.5　创建广播接收器的窗口

可以看到，这里我们将广播接收器命名为 BootCompleteReceiver， Exported 属性表示是否允许这个广播接收器接收本程序以外的广播，Enabled 属性表示是否启用这个广播接收器。勾选这两个属性，点击 Finish 完成创建。

然后修改 BootCompleteReceiver 中的代码，如下所示：

```
public class BootCompleteReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Boot Complete", Toast.LENGTH_LONG).show();
    }

}
```

代码非常简单，我们只是在 onReceive()方法中使用 Toast 弹出一段提示信息。

另外，静态的广播接收器一定要在 AndroidManifest.xml 文件中注册才可以使用，不过由于我们是使用 Android Studio 的快捷方式创建的广播接收器，因此注册这一步已经被自动完成了。打开 AndroidManifest.xml 文件瞧一瞧，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest">

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
```

```
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".BootCompleteReceiver"
            android:enabled="true"
            android:exported="true">
    </receiver>
    </application>

</manifest>
```

可以看到，<application>标签内出现了一个新的标签<receiver>，所有静态的广播接收器都是在这里进行注册的。它的用法其实和<activity>标签非常相似，也是通过 android:name 来指定具体注册哪一个广播接收器，而 enabled 和 exported 属性则是根据我们刚才勾选的状态自动生成的。

不过目前 BootCompleteReceiver 还是不能接收到开机广播的，我们还需要对 AndroidManifest.xml 文件进行修改才行，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest">

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".BootCompleteReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

由于 Android 系统启动完成后会发出一条值为 android.intent.action.BOOT_COMPLETED 的广播，因此我们在<intent-filter>标签里添加了相应的 action。另外，监听系统开机广播也是需要声明权限的，可以看到，我们使用<uses-permission>标签又加入了一条 android.permission.RECEIVE_BOOT_COMPLETED 权限。

现在重新运行程序后，我们的程序就已经可以接收开机广播了。将模拟器关闭并重新启动，在启动完成之后就会收到开机广播，如图 5.6 所示。



图 5.6 接收系统开机广播

到目前为止，我们在广播接收器的 onReceive()方法中都只是简单地使用 Toast 提示了一段文本信息，当你真正在项目中使用到它的时候，就可以在里面编写自己的逻辑。需要注意的是，不要在 onReceive()方法中添加过多的逻辑或者进行任何的耗时操作，因为在广播接收器中是不允许开启线程的，当 onReceive()方法运行了较长时间而没有结束时，程序就会报错。因此广播接收器更多的是扮演一种打开程序其他组件的角色，比如创建一条状态栏通知，或者启动一个服务等，这几个概念我们会在后面的章节中学到。

5.3 发送自定义广播

现在你已经学会了通过广播接收器来接收系统广播，接下来我们就要学习一下如何在应用程序中发送自定义的广播。前面已经介绍过了，广播主要分为两种类型：标准广播和有序广播，在本节中我们就将通过实践的方式来看一下这两种广播具体的区别。

5.3.1 发送标准广播

在发送广播之前，我们还是需要先定义一个广播接收器来准备接收此广播才行，不然发出去也是白发。因此新建一个 MyBroadcastReceiver，代码如下所示：

```
public class MyBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in MyBroadcastReceiver", Toast.LENGTH_
            SHORT).show();
    }

}
```

这里当 MyBroadcastReceiver 收到自定义的广播时，就会弹出“received in MyBroadcast-Receiver”的提示。然后在 AndroidManifest.xml 中对这个广播接收器进行修改：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest">
    ...
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".MyBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="com.example.broadcasttest.MY_BROADCAST"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

可以看到，这里让 MyBroadcastReceiver 接收一条值为 `com.example.broadcasttest.MY_BROADCAST` 的广播，因此待会儿在发送广播的时候，我们就需要发出这样的一条广播。

接下来修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send Broadcast"
        />

</LinearLayout>
```

这里在布局文件中定义了一个按钮，用于作为发送广播的触发点。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
            Button button = (Button) findViewById(R.id.button);
            button.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent intent = new
                    Intent("com.example.broadcasttest.MY_BROADCAST");
                    sendBroadcast(intent);
                }
            });
            ...
        }

        ...

    }
```

可以看到，我们在按钮的点击事件里面加入了发送自定义广播的逻辑。首先构建出了一个 `Intent` 对象，并把要发送的广播的值传入，然后调用了 Context 的 `sendBroadcast()`方法将广播发送出去，这样所有监听 `com.example.broadcasttest.MY_BROADCAST` 这条广播的广播接收器就会收到消息。此时发出去的广播就是一条标准广播。

重新运行程序，并点击一下 Send Broadcast 按钮，效果如图 5.7 所示。

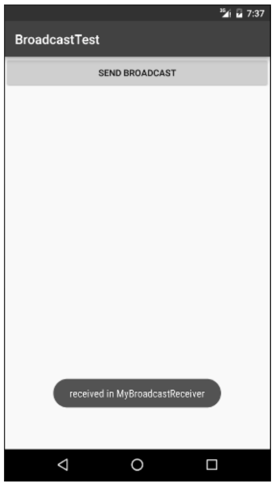


图 5.7 接收到自定义广播

这样我们就成功完成了发送自定义广播的功能。另外，由于广播是使用 Intent 进行传递的，因此你还可以在 Intent 中携带一些数据传递给广播接收器。

5.3.2 发送有序广播

广播是一种可以跨进程的通信方式，这一点从前面接收系统广播的时候就可以看出来了。因此在我们应用程序内发出的广播，其他的应用程序应该也是可以收到的。为了验证这一点，我们

需要再新建一个 BroadcastTest2 项目，点击 Android Studio 导航栏→File→New→New Project 进行创建。

将项目创建好之后，还需要在这个项目下定义一个广播接收器，用于接收上一小节中的自定义广播。新建 AnotherBroadcastReceiver，代码如下所示：

```
public class AnotherBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in AnotherBroadcastReceiver",
            Toast.LENGTH_SHORT).show();
    }

}
```

这里仍然是在广播接收器的 onReceive()方法中弹出了一段文本信息。然后在 AndroidManifest.xml 中对这个广播接收器进行修改，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest2">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".AnotherBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="com.example.broadcasttest.MY_BROADCAST" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

可以看到，AnotherBroadcastReceiver 同样接收的是 com.example.broadcasttest.MY_BROADCAST 这条广播。现在运行 BroadcastTest2 项目将这个程序安装到模拟器上，然后重新回到 BroadcastTest 项目的主界面，并点击一下 Send Broadcast 按钮，就会分别弹出两次提示信息，如图 5.8 所示。

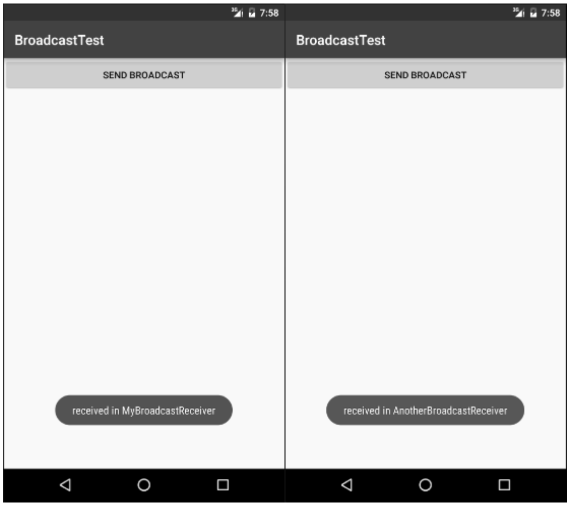


图 5.8 两个程序中都接收到自定义广播

这样就强有力地证明了，我们的应用程序发出的广播是可以被其他的应用程序接收到的。

不过到目前为止，程序里发出的都还是标准广播，现在我们来尝试一下发送有序广播。重新回到 BroadcastTest 项目，然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new
                    Intent("com.example.broadcasttest.MY_BROADCAST");
                sendOrderedBroadcast(intent, null);
            }
        });
        ...
    }

    ...

}
```

可以看到，发送有序广播只需要改动一行代码，即将 sendBroadcast()方法改成 sendOrderedBroadcast()方法。sendOrderedBroadcast()方法接收两个参数，第一个参数仍然是

Intent，第二个参数是一个与权限相关的字符串，这里传入 null 就行了。现在重新运行程序，并点击 Send Broadcast 按钮，你会发现，两个应用程序仍然都可以接收到这条广播。

看上去好像和标准广播没什么区别嘛，不过别忘了，这个时候的广播接收器是有先后顺序的，而且前面的广播接收器还可以将广播截断，以阻止其继续传播。

那么该如何设定广播接收器的先后顺序呢？当然是在注册的时候进行设定的了，修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest2">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".MyBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter android:priority="100">
                <action android:name="com.example.broadcasttest.MY_BROADCAST" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

可以看到，我们通过 android:priority 属性给广播接收器设置了优先级，优先级比较高的广播接收器就可以先收到广播。这里将 MyBroadcastReceiver 的优先级设成了 100，以保证它一定会在 AnotherBroadcastReceiver 之前收到广播。

既然已经获得了接收广播的优先权，那么 MyBroadcastReceiver 就可以选择是否允许广播继续传递了。修改 MyBroadcastReceiver 中的代码，如下所示：

```
public class MyBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in MyBroadcastReceiver",
            Toast.LENGTH_SHORT).show();
        abortBroadcast();
    }

}
```

如果在 onReceive()方法中调用了 abortBroadcast()方法，就表示将这条广播截断，后面

的广播接收器将无法再接收到这条广播。现在重新运行程序，并点击一下 Send Broadcast 按钮，你会发现，只有 MyBroadcastReceiver 中的 Toast 信息能够弹出，说明这条广播经过 MyBroadcast-Receiver 之后确实是终止传递了。

5.4　使用本地广播

前面我们发送和接收的广播全部属于系统全局广播，即发出的广播可以被其他任何应用程序接收到，并且我们也可以接收来自于其他任何应用程序的广播。这样就很容易引起安全性的问题，比如说我们发送的一些携带关键性数据的广播有可能被其他的应用程序截获，或者其他的程序不停地向我们的广播接收器里发送各种垃圾广播。

为了能够简单地解决广播的安全性问题，Android 引入了一套本地广播机制，使用这个机制发出的广播只能够在应用程序的内部进行传递，并且广播接收器也只能接收来自本应用程序发出的广播，这样所有的安全性问题就都不存在了。

本地广播的用法并不复杂，主要就是使用了一个 LocalBroadcastManager 来对广播进行管理，并提供了发送广播和注册广播接收器的方法。下面我们就通过具体的实例来尝试一下它的用法，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private IntentFilter intentFilter;

    private LocalReceiver localReceiver;

    private LocalBroadcastManager localBroadcastManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        localBroadcastManager = LocalBroadcastManager.getInstance(this); // 获取实例
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent("com.example.broadcasttest.LOCAL_
                    BROADCAST");
                localBroadcastManager.sendBroadcast(intent); // 发送本地广播
            }
        });
        intentFilter = new IntentFilter();
        intentFilter.addAction("com.example.broadcasttest.LOCAL_BROADCAST");
        localReceiver = new LocalReceiver();
        localBroadcastManager.registerReceiver(localReceiver, intentFilter); // 注
        册本地广播监听器
    }
```

```
    @Override
    protected void onDestroy() {
        super.onDestroy();
        localBroadcastManager.unregisterReceiver(localReceiver);
    }

    class LocalReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(context, "received local broadcast", Toast.LENGTH_SHORT).
               show();
        }

    }

}
```

有没有感觉这些代码很熟悉？没错，其实这基本上就和我们前面所学的动态注册广播接收器以及发送广播的代码是一样的。只不过现在首先是通过 LocalBroadcastManager 的 `getInstance()`方法得到了它的一个实例，然后在注册广播接收器的时候调用的是 LocalBroadcastManager 的 `registerReceiver()`方法，在发送广播的时候调用的是 LocalBroadcastManager 的 `sendBroadcast()`方法，仅此而已。这里我们在按钮的点击事件里面发出了一条 `com.example.broadcasttest.LOCAL_BROADCAST` 广播，然后在 LocalReceiver 里去接收这条广播。重新运行程序，并点击 Send Broadcast 按钮，效果如图 5.9 所示。

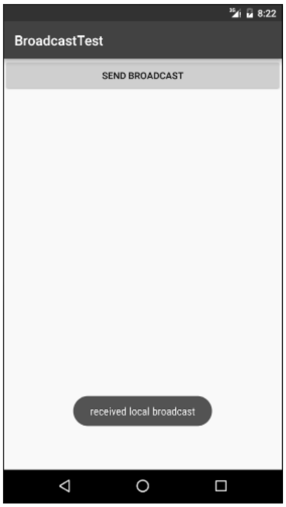


图 5.9　接收到本地广播

可以看到，LocalReceiver 成功接收到了这条本地广播，并通过 Toast 提示了出来。如果你还有兴趣进行实验，可以尝试在 BroadcastTest2 中也去接收 `com.example.broadcasttest.`

LOCAL_BROADCAST 这条广播。答案是显而易见的，肯定无法收到，因为这条广播只会在 BroadcastTest 程序内传播。

另外还有一点需要说明，本地广播是无法通过静态注册的方式来接收的。其实这也完全可以理解，因为静态注册主要就是为了让程序在未启动的情况下也能收到广播，而发送本地广播时，我们的程序肯定是已经启动了，因此也完全不需要使用静态注册的功能。

最后我们再来盘点一下使用本地广播的几点优势吧。

- 可以明确地知道正在发送的广播不会离开我们的程序，因此不必担心机密数据泄漏。
- 其他的程序无法将广播发送到我们程序的内部，因此不需要担心会有安全漏洞的隐患。
- 发送本地广播比发送系统全局广播将会更加高效。

5.5　广播的最佳实践——实现强制下线功能

本章的内容不是非常多，因此相信你也一定学得很轻松吧。现在我们就准备通过一个完整例子的实践，来综合运用一下本章中所学到的知识。

强制下线功能应该算是比较常见的了，很多的应用程序都具备这个功能，比如你的 QQ 号在别处登录了，就会将你强制挤下线。其实实现强制下线功能的思路也比较简单，只需要在界面上弹出一个对话框，让用户无法进行任何其他操作，必须要点击对话框中的确定按钮，然后回到登录界面即可。可是这样就存在着一个问题，因为当我们被通知需要强制下线时可能正处于任何一个界面，难道需要在每个界面上都编写一个弹出对话框的逻辑？如果你真的这么想，那思维就偏远了，我们完全可以借助本章中所学的广播知识，来非常轻松地实现这一功能。新建一个 BroadcastBestPractice 项目，然后开始动手吧。

强制下线功能需要先关闭掉所有的活动，然后回到登录界面。如果你的反应足够快的话，应该会想到我们在第 2 章的最佳实践部分早就已经实现过关闭所有活动的功能了，因此这里只需要使用同样的方案即可。先创建一个 ActivityCollector 类用于管理所有的活动，代码如下所示：

```
public class ActivityCollector {

    public static List<Activity> activities = new ArrayList<>();

    public static void addActivity(Activity activity) {
        activities.add(activity);
    }

    public static void removeActivity(Activity activity) {
        activities.remove(activity);
    }

    public static void finishAll() {
        for (Activity activity : activities) {
            if (!activity.isFinishing()) {
                activity.finish();
```

```
            }
        }
        activities.clear();
    }

}
```

然后创建 BaseActivity 类作为所有活动的父类，代码如下所示：

```
public class BaseActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityCollector.addActivity(this);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        ActivityCollector.removeActivity(this);
    }

}
```

以上代码都是直接拿之前写好的内容，非常开心。不过从这里开始，就要靠我们自己去动手实现了。首先需要创建一个登录界面的活动，新建 LoginActivity，并让 Android Studio 帮我们自动生成相应的布局文件。然后编辑布局文件 activity_login.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="60dp">
        <TextView
            android:layout_width="90dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:textSize="18sp"
            android:text="Account:" />

        <EditText
            android:id="@+id/account"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:layout_gravity="center_vertical" />
    </LinearLayout>

    <LinearLayout
        android:orientation="horizontal"
```

```
        android:layout_width="match_parent"
        android:layout_height="60dp">
        <TextView
            android:layout_width="90dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:textSize="18sp"
            android:text="Password:" />

        <EditText
            android:id="@+id/password"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:layout_gravity="center_vertical"
            android:inputType="textPassword" />
    </LinearLayout>

    <Button
        android:id="@+id/login"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:text="Login" />

</LinearLayout>
```

这里我们使用 LinearLayout 编写出了一个登录布局，最外层是一个纵向的 LinearLayout，里面包含了 3 行直接子元素。第一行是一个横向 LinearLayout，用于输入账号信息；第二行也是一个横向的 LinearLayout，用于输入密码信息；第三行是一个登录按钮。这个布局文件里面用到的全部都是我们之前学过的内容，相信你理解起来应该不会费劲。

接下来修改 LoginActivity 中的代码，如下所示：

```
public class LoginActivity extends BaseActivity {

    private EditText accountEdit;

    private EditText passwordEdit;

    private Button login;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        accountEdit = (EditText) findViewById(R.id.account);
        passwordEdit = (EditText) findViewById(R.id.password);
        login = (Button) findViewById(R.id.login);
        login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String account = accountEdit.getText().toString();
```

```
                String password = passwordEdit.getText().toString();
                // 如果账号是admin且密码是123456，就认为登录成功
                if (account.equals("admin") && password.equals("123456")) {
                    Intent intent = new Intent(LoginActivity.this, MainActivity.
                        class);
                    startActivity(intent);
                    finish();
                } else {
                    Toast.makeText(LoginActivity.this, "account or password is
                       invalid", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }

}
```

这里我们模拟了一个非常简单的登录功能。首先要将 LoginActivity 的继承结构改成继承自 BaseActivity，然后调用 `findViewById()`方法分别获取到账号输入框、密码输入框以及登录按钮的实例。接着在登录按钮的点击事件里面对输入的账号和密码进行判断，如果账号是 admin 并且密码是 123456，就认为登录成功并跳转到 MainActivity，否则就提示用户账号或密码错误。

因此，你就可以将 MainActivity 理解成是登录成功后进入的程序主界面了，这里我们并不需要在主界面里提供什么花哨的功能，只需要加入强制下线功能就可以了，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/force_offline"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send force offline broadcast" />

</LinearLayout>
```

非常简单，只有一个按钮而已，用于触发强制下线功能。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends BaseActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button forceOffline = (Button) findViewById(R.id.force_offline);
        forceOffline.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                Intent intent = new Intent("com.example.broadcastbestpractice.
                    FORCE_OFFLINE");
                sendBroadcast(intent);
            }
        });
    }

}
```

同样非常简单，不过这里有个重点，我们在按钮的点击事件里面发送了一条广播，广播的值为 `com.example.broadcastbestpractice.FORCE_OFFLINE`，这条广播就是用于通知程序强制用户下线的。也就是说强制用户下线的逻辑并不是写在 MainActivity 里的，而是应该写在接收这条广播的广播接收器里面，这样强制下线的功能就不会依附于任何的界面，不管是在程序的任何地方，只需要发出这样一条广播，就可以完成强制下线的操作了。

那么毫无疑问，接下来我们就需要创建一个广播接收器来接收这条强制下线广播，唯一的问题就是，应该在哪里创建呢？由于广播接收器里面需要弹出一个对话框来阻塞用户的正常操作，但如果创建的是一个静态注册的广播接收器，是没有办法在 `onReceive()`方法里弹出对话框这样的 UI 控件的，而我们显然也不可能在每个活动中都去注册一个动态的广播接收器。

那么到底应该怎么办呢？答案其实很明显，只需要在 BaseActivity 中动态注册一个广播接收器就可以了，因为所有的活动都是继承自 BaseActivity 的。

修改 BaseActivity 中的代码，如下所示：

```
public class BaseActivity extends AppCompatActivity {

    private ForceOfflineReceiver receiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityCollector.addActivity(this);
    }

    @Override
    protected void onResume() {
        super.onResume();
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction("com.example.broadcastbestpractice.FORCE_OFFLINE");
        receiver = new ForceOfflineReceiver();
        registerReceiver(receiver, intentFilter);
    }

    @Override
    protected void onPause() {
        super.onPause();
        if (receiver != null) {
            unregisterReceiver(receiver);
```

```
            receiver = null;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        ActivityCollector.removeActivity(this);
    }

    class ForceOfflineReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(final Context context, Intent intent) {
            AlertDialog.Builder builder = new AlertDialog.Builder(context);
            builder.setTitle("Warning");
            builder.setMessage("You are forced to be offline. Please try to login
                again.");
            builder.setCancelable(false);
            builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    ActivityCollector.finishAll(); // 销毁所有活动
                    Intent intent = new Intent(context, LoginActivity.class);
                    context.startActivity(intent); // 重新启动 LoginActivity
                }
            });
            builder.show();
        }

    }

}
```

先来看一下 ForceOfflineReceiver 中的代码，这次 `onReceive()`方法里可不再是仅仅弹出一个 Toast 了，而是加入了较多的代码，那我们就来仔细地看看吧。首先肯定是使用 `AlertDialog.Builder` 来构建一个对话框，注意这里一定要调用 `setCancelable()`方法将对话框设为不可取消，否则用户按一下 Back 键就可以关闭对话框继续使用程序了。然后使用 `setPositiveButton()`方法来给对话框注册确定按钮，当用户点击了确定按钮时，就调用 ActivityCollector 的 `finishAll()`方法来销毁掉所有活动，并重新启动 LoginActivity 这个活动。

再来看一下我们是怎么注册 ForceOfflineReceiver 这个广播接收器的，可以看到，这里重写了 `onResume()`和 `onPause()`这两个生命周期函数，然后分别在这两个方法里注册和取消注册了 ForceOfflineReceiver。

那么为什么要这样写呢？之前不都是在 `onCreate()`和 `onDestroy()`方法里来注册和取消注册广播接收器的么？这是因为我们始终需要保证只有处于栈顶的活动才能接收到这条强制下线广播，非栈顶的活动不应该也没有必要去接收这条广播，所以写在 `onResume()`和 `onPause()`方法里就可以很好地解决这个问题，当一个活动失去栈顶位置时就会自动取消广播接收器的注册。

这样的话，所有强制下线的逻辑就已经完成了，接下来我们还需要对 AndroidManifest.xml 文件进行修改，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcastbestpractice">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
        </activity>
        <activity android:name=".LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

这里只需要对一处代码进行修改，就是将主活动设置为 LoginActivity 而不再是 MainActivity，因为你肯定不希望用户在没登录的情况下就能直接进入到程序主界面吧？

好了，现在来尝试运行一下程序吧，首先会进入到登录界面，并可以在这里输入账号和密码，如图 5.10 所示。



图 5.10　登录界面

如果输入的账号是 admin，密码是 123456，点击登录按钮就会进入到程序的主界面，如图 5.11 所示。这时点击一下发送广播的按钮，就会发出一条强制下线的广播，ForceOfflineReceiver 里收到这条广播后会弹出一个对话框提示用户已被强制下线，如图 5.12 所示。

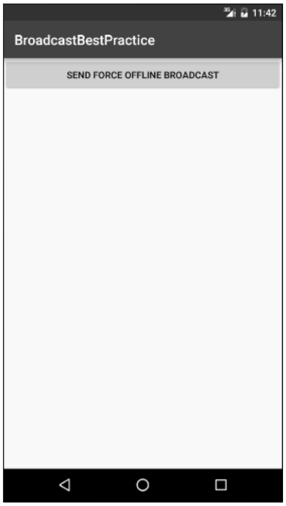


图 5.11　主界面

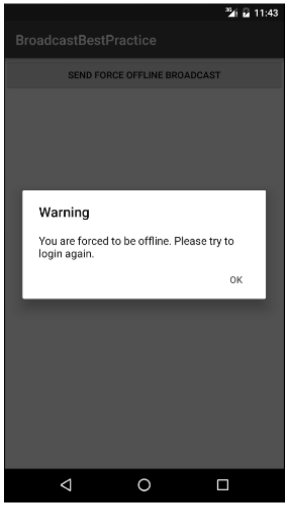


图 5.12　强制下线提示

这时用户将无法再对界面的任何元素进行操作，只能点击确定按钮，然后会重新回到登录界面。这样，强制下线功能就已经完整地实现了。

结束了本章的最佳实践部分，接下来我们要进入一个特殊的环节。相信你一定也知道，几乎所有出色的项目都不会是由一个人单枪匹马完成的，而是由一个团队共同合作开发完成的。这个时候多人之间代码同步的问题就显得异常重要，因此版本控制工具也就应运而生了。常见的版本控制工具主要有 svn 和 Git，本书中将会对 Git 的使用方法进行全面的讲解，并且讲解的内容是穿插于一些章节当中的。那么今天，我们就先来看一看关于 Git 最基本的用法。

5.6　Git 时间——初识版本控制工具

Git 是一个开源的分布式版本控制工具，它的开发者就是鼎鼎大名的 Linux 操作系统的作者 Linus Torvalds。Git 被开发出来的初衷是为了更好地管理 Linux 内核，而现在却早已被广泛应用于全球各种大中小型的项目中。今天是我们关于 Git 的第一堂课，主要是讲解一下它最基本的用法，那么就从安装 Git 开始吧。

5.6.1　安装 Git

由于 Git 和 Linux 操作系统都是同一个作者，因此不用我说，你也应该猜到 Git 在 Linux 上的

安装是最简单方便的。比如你使用的是 Ubuntu 系统，只需要打开 shell 界面，并输入：

```
sudo apt-get install git-core
```

按下回车后输入密码，即可完成 Git 的安装。

不过我相信你更有可能使用的还是 Windows 操作系统，因此本小节的重点是教会你如何在 Windows 上安装 Git。不同于 Linux，Windows 上可无法通过一行命令就完成安装了，我们需要先把 Git 的安装包下载下来。访问网址 https://git-for-windows.github.io/，可以看到如图 5.13 所示的页面。



图 5.13　git for windows 主页

目前最新的 git for windows 版本是 2.8.1，我就准备使用这一版本了，如果你下载的时候发现又有新的版本，可以尝试一下最新版本的 Git。点击 Download 按钮可以开始下载，下载完成后双击安装包进行安装，之后一直点击“下一步”就可以完成安装了。

5.6.2　创建代码仓库

虽然在 Windows 上安装的 Git 是可以在图形界面上进行操作的，并且 Android Studio 也支持以图形化的形式操作 Git，但是这里我并不建议你这样做，因为 Git 的各种命令才是你应该掌握的核心技能，不管你是在哪个操作系统中，使用命令来操作 Git 肯定都是通用的。而图形化的操作应该是在你能熟练掌握命令用法的前提下，进一步提升你工作效率的手段。

那么我们现在就来尝试一下如何通过命令来使用 Git。如果你使用的是 Linux 系统，就先打开 shell 界面，如果使用的是 Windows 系统，就从开始里找到 Git Bash 并打开。

首先应该配置一下你的身份，这样在提交代码的时候 Git 就可以知道是谁提交的了，命令如下所示：

```
git config --global user.name "Tony"
git config --global user.email "tony@gmail.com"
```

配置完成后你还可以使用同样的命令来查看是否配置成功，只需要将最后的名字和邮箱地址去掉即可，如图 5.14 所示。

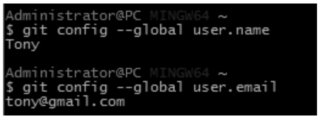


图 5.14　查看 git 用户名和邮箱

然后我们就可以开始创建代码仓库了，仓库（Repository）是用于保存版本管理所需信息的地方，所有本地提交的代码都会被提交到代码仓库中，如果有需要还可以再推送到远程仓库中。

这里我们尝试着给 BroadcastBestPractice 项目建立一个代码仓库。先进入到 BroadcastBest-Practice 项目的目录下面，如图 5.15 所示。

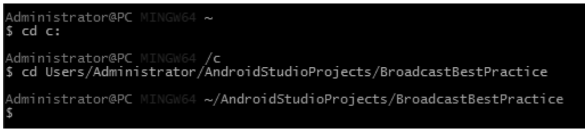


图 5.15　切换到 BroadcastBestPractice 项目目录下

然后在这个目录下面输入如下命令：

```
git init
```

很简单吧！只需要一行命令就可以完成创建代码仓库的操作，如图 5.16 所示。

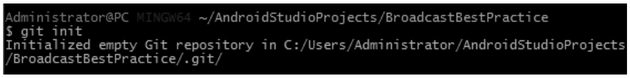


图 5.16　创建代码仓库

仓库创建完成后，会在 BroadcastBestPractice 项目的根目录下生成一个隐藏的.git 文件夹，这个文件夹就是用来记录本地所有的 Git 操作的，可以通过 `ls -al` 命令来查看一下，如图 5.17 所示。

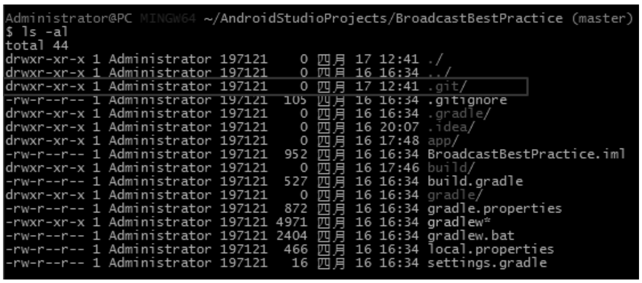


图 5.17　查看.git 文件

如果你想要删除本地仓库，只需要删除这个文件夹就行了。

5.6.3 提交本地代码

代码仓库建立完之后就可以提交代码了，其实提交代码的方法也非常简单，只需要使用 add 和 commit 命令就可以了。add 用于把想要提交的代码先添加进来，而 commit 则是真正地去执行提交操作。比如我们想添加 build.gradle 文件，就可以输入如下命令：

```
git add build.gradle
```

这是添加单个文件的方法，那如果我们想添加某个目录呢？其实只需要在 add 后面加上目录名就可以了。比如将整个 app 目录下的所有文件都进行添加，就可以输入如下命令：

```
git add app
```

可是这样一个个地添加感觉还是有些复杂，有没有什么办法可以一次性就把所有的文件都添加好呢？当然可以，只需要在 add 的后面加上一个点，就表示添加所有的文件了，命令如下所示：

```
git add .
```

现在 BroadcastBestPractice 项目下所有的文件都已经添加好了，我们可以来提交一下了，输入如下命令：

```
git commit -m "First commit."
```

注意，在 commit 命令的后面，我们一定要通过-m 参数来加上提交的描述信息，没有描述信息的提交被认为是不合法的。这样所有的代码就已经成功提交了！

好了，关于 Git 的内容，今天我们就学到这里，虽然内容并不多，但是你已经将 Git 最基本的用法都掌握了，不是吗？在本书后面的章节，还会穿插一些 Git 的讲解，到时候你将学会更多关于 Git 的使用技巧，现在就让我们来总结一下吧。

5.7 小结与点评

本章中我们主要是对 Android 的广播机制进行了深入的研究，不仅了解了广播的理论知识，还掌握了接收广播、发送自定义广播以及本地广播的使用方法。广播接收器属于 Android 四大组件之一，在不知不觉中你已经掌握了四大组件中的两个了。

在最佳实践环节中你一定也收获了不少，不仅运用到了本章所学的广播知识，还将前面章节所学到的技巧综合运用到了一起。经过这个例子之后，相信你对所涉及的每个知识点都有了更深一层的认识。另外，本章还添加了一个最最特殊的环节，即 Git 时间。在这个环节中，我们对 Git 这个版本控制工具进行了初步的学习，后面还会学习关于它的更多内容。

下一章我们本应该继续学习 Android 四大组件中的内容提供器，不过由于学习内容提供器之前需要先掌握 Android 中的持久化技术，因此下一章我们就先对这一主题展开讨论。

第 6 章

数据存储全方案——详解持久化技术

任何一个应用程序，其实说白了就是在不停地和数据打交道，我们聊 QQ、看新闻、刷微博，所关心的都是里面的数据，没有数据的应用程序就变成了一个空壳子，对用户来说没有任何实际用途。那么这些数据都是从哪来的呢？现在多数的数据基本都是由用户产生的，比如你发微博、评论新闻，其实都是在产生数据。

而我们前面章节所编写的众多例子中也有用到各种各样的数据，例如第 3 章最佳实践部分在聊天界面编写的聊天内容，第 5 章最佳实践部分在登录界面输入的账号和密码。这些数据都有一个共同点，即它们都属于瞬时数据。那么什么是瞬时数据呢？就是指那些存储在内存当中，有可能会因为程序关闭或其他原因导致内存被回收而丢失的数据。这对于一些关键性的数据信息来说是绝对不能容忍的，谁都不希望自己刚发出去的一条微博，刷新一下就没了吧。那么怎样才能保证一些关键性的数据不会丢失呢？这就需要用到数据持久化技术了。

6.1 持久化技术简介

数据持久化就是指将那些内存中的瞬时数据保存到存储设备中，保证即使在手机或电脑关机的情况下，这些数据仍然不会丢失。保存在内存中的数据是处于瞬时状态的，而保存在存储设备中的数据是处于持久状态的，持久化技术则提供了一种机制可以让数据在瞬时状态和持久状态之间进行转换。

持久化技术被广泛应用于各种程序设计的领域当中，而本书中要探讨的自然是 Android 中的数据持久化技术。Android 系统中主要提供了 3 种方式用于简单地实现数据持久化功能，即文件存储、SharedPreferences 存储以及数据库存储。当然，除了这 3 种方式之外，你还可以将数据保存在手机的 SD 卡中，不过使用文件、SharedPreferences 或数据库来保存数据会相对更简单一些，而且比起将数据保存在 SD 卡中会更加地安全。

那么下面我就将对这 3 种数据持久化的方式一一进行详细的讲解。

6.2 文件存储

文件存储是 Android 中最基本的一种数据存储方式，它不对存储的内容进行任何的格式化处理，所有数据都是原封不动地保存到文件当中的，因而它比较适合用于存储一些简单的文本数据或二进制数据。如果你想使用文件存储的方式来保存一些较为复杂的文本数据，就需要定义一套自己的格式规范，这样可以方便之后将数据从文件中重新解析出来。

那么首先我们就来看一看，Android 中是如何通过文件来保存数据的。

6.2.1 将数据存储到文件中

Context 类中提供了一个 openFileOutput()方法，可以用于将数据存储到指定的文件中。这个方法接收两个参数，第一个参数是文件名，在文件创建的时候使用的就是这个名称，注意这里指定的文件名不可以包含路径，因为所有的文件都是默认存储到/data/data/<package name>/files/目录下的。第二个参数是文件的操作模式，主要有两种模式可选，MODE_PRIVATE 和 MODE_APPEND。其中 MODE_PRIVATE 是默认的操作模式，表示当指定同样文件名的时候，所写入的内容将会覆盖原文件中的内容，而 MODE_APPEND 则表示如果该文件已存在，就往文件里面追加内容，不存在就创建新文件。其实文件的操作模式本来还有另外两种：MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE，这两种模式表示允许其他的应用程序对我们程序中的文件进行读写操作，不过由于这两种模式过于危险，很容易引起应用的安全性漏洞，已在 Android 4.2 版本中被废弃。

openFileOutput ()方法返回的是一个 FileOutputStream 对象，得到了这个对象之后就可以使用 Java 流的方式将数据写入到文件中了。以下是一段简单的代码示例，展示了如何将一段文本内容保存到文件中：

```
public void save() {
    String data = "Data to save";
    FileOutputStream out = null;
    BufferedWriter writer = null;
    try {
        out = openFileOutput("data", Context.MODE_PRIVATE);
        writer = new BufferedWriter(new OutputStreamWriter(out));
        writer.write(data);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (writer != null) {
                writer.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

如果你已经比较熟悉 Java 流了，理解上面的代码一定轻而易举吧。这里通过 `openFileOutput()`方法能够得到一个 `FileOutputStream` 对象，然后再借助它构建出一个 `OutputStreamWriter` 对象，接着再使用 `OutputStreamWriter` 构建出一个 `BufferedWriter` 对象，这样你就可以通过 `BufferedWriter` 来将文本内容写入到文件中了。

下面我们就编写一个完整的例子，借此学习一下如何在 Android 项目中使用文件存储的技术。首先创建一个 FilePersistenceTest 项目，并修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText
        android:id="@+id/edit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Type something here"
        />

</LinearLayout>
```

这里只是在布局中加入了一个 EditText，用于输入文本内容。其实现在你就可以运行一下程序了，界面上肯定会有一个文本输入框。然后在文本输入框中随意输入点什么内容，再按下 Back 键，这时输入的内容肯定就已经丢失了，因为它只是瞬时数据，在活动被销毁后就会被回收。而这里我们要做的，就是在数据被回收之前，将它存储到文件当中。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private EditText edit;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        edit = (EditText) findViewById(R.id.edit);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        String inputText = edit.getText().toString();
        save(inputText);
    }

    public void save(String inputText) {
        FileOutputStream out = null;
        BufferedWriter writer = null;
        try {
```

```
                out = openFileOutput("data", Context.MODE_PRIVATE);
                writer = new BufferedWriter(new OutputStreamWriter(out));
                writer.write(inputText);
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
                    if (writer != null) {
                        writer.close();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

    }
```

可以看到，首先我们在 onCreate()方法中获取了 EditText 的实例，然后重写了 onDestroy()方法，这样就可以保证在活动销毁之前一定会调用这个方法。在 onDestroy()方法中我们获取了 EditText 中输入的内容，并调用 save()方法把输入的内容存储到文件中，文件命名为 data。save()方法中的代码和之前的示例基本相同，这里就不再做解释了。现在重新运行一下程序，并在 EditText 中输入一些内容，如图 6.1 所示。

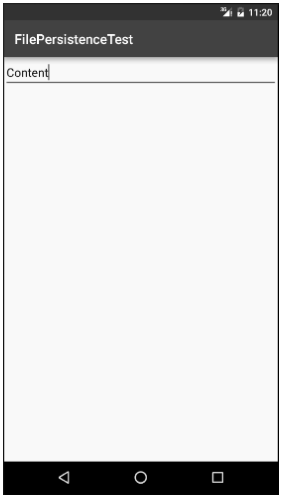


图 6.1　在 EditText 中随意输入点内容

然后按下 Back 键关闭程序，这时我们输入的内容就已经保存到文件中了。那么如何才能证实数据确实已经保存成功了呢？我们可以借助 Android Device Monitor 工具来查看一下。点击 Android Studio 导航栏中的 Tools→Android，会看到如图 6.2 所示的工具列表。

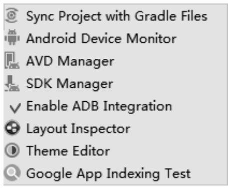


图 6.2　Android 工具列表

点击 Android Device Monitor 就可以打开 Android Device Monitor 工具了，然后进入 File Explorer 标签页，在这里找到/data/data/com.example.filepersistencetest/files/目录，可以看到生成了一个 data 文件，如图 6.3 所示。（注：Android 7.0 系统的模拟器可能无法正常查看 File Explorer 中的内容，这或许是新版模拟器的一个 bug，可能会在未来的版本中修复。如果你遇到了这种情况，创建一个 Android 6.0 系统的模拟器即可解决。）

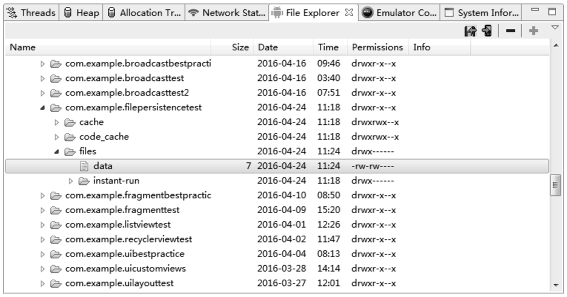


图 6.3　生成的 data 文件

然后点击图 6.4 中左边的按钮可以将这个文件导出到电脑上。



图 6.4　导入导出按钮

使用记事本打开这个文件，里面的内容如图 6.5 所示。



图 6.5　data 文件中的内容

这样就证实了，在 EditText 中输入的内容确实已经成功保存到文件中了。

不过只是成功将数据保存下来还不够，我们还需要想办法在下次启动程序的时候让这些数据

能够还原到 EditText 中，因此接下来我们就要学习一下如何从文件中读取数据。

6.2.2 从文件中读取数据

类似于将数据存储到文件中，Context 类中还提供了一个 openFileInput()方法，用于从文件中读取数据。这个方法要比 openFileOutput()简单一些，它只接收一个参数，即要读取的文件名，然后系统会自动到/data/data/<package name>/files/目录下去加载这个文件，并返回一个 FileInputStream 对象，得到了这个对象之后再通过 Java 流的方式就可以将数据读取出来了。

以下是一段简单的代码示例，展示了如何从文件中读取文本数据：

```
public String load() {
    FileInputStream in = null;
    BufferedReader reader = null;
    StringBuilder content = new StringBuilder();
    try {
        in = openFileInput("data");
        reader = new BufferedReader(new InputStreamReader(in));
        String line = "";
        while ((line = reader.readLine()) != null) {
            content.append(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return content.toString();
}
```

在这段代码中，首先通过 openFileInput()方法获取到了一个 FileInputStream 对象，然后借助它又构建出了一个 InputStreamReader 对象，接着再使用 InputStreamReader 构建出一个 BufferedReader 对象，这样我们就可以通过 BufferedReader 进行一行行地读取，把文件中所有的文本内容全部读取出来，并存放在一个 StringBuilder 对象中，最后将读取到的内容返回就可以了。

了解了从文件中读取数据的方法，那么我们就来继续完善上一小节中的例子，使得重新启动程序时 EditText 中能够保留我们上次输入的内容。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private EditText edit;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        edit = (EditText) findViewById(R.id.edit);
        String inputText = load();
        if (!TextUtils.isEmpty(inputText)) {
            edit.setText(inputText);
            edit.setSelection(inputText.length());
            Toast.makeText(this, "Restoring succeeded", Toast.LENGTH_SHORT).show();
        }
    }

    ...

    public String load() {
        FileInputStream in = null;
        BufferedReader reader = null;
        StringBuilder content = new StringBuilder();
        try {
            in = openFileInput("data");
            reader = new BufferedReader(new InputStreamReader(in));
            String line = "";
            while ((line = reader.readLine()) != null) {
                content.append(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return content.toString();
    }

}
```

可以看到，这里的思路非常简单，在 onCreate()方法中调用 load()方法来读取文件中存储的文本内容，如果读到的内容不为 null，就调用 EditText 的 setText()方法将内容填充到 EditText 里，并调用 setSelection()方法将输入光标移动到文本的末尾位置以便于继续输入，然后弹出一句还原成功的提示。load()方法中的细节我们在前面已经讲过，这里就不再赘述了。

注意，上述代码在对字符串进行非空判断的时候使用了 TextUtils.isEmpty()方法，这是一个非常好用的方法，它可以一次性进行两种空值的判断。当传入的字符串等于 null 或者等于空字符串的时候，这个方法都会返回 true，从而使得我们不需要先单独判断这两种空值再使用逻辑运算符连接起来了。

现在重新运行一下程序，刚才保存的 Content 字符串肯定会被填充到 EditText 中，然后编写一点其他的内容，比如在 EditText 中输入 Hello，接着按下 Back 键退出程序，再重新启动程序，这时刚才输入的内容并不会丢失，而是还原到了 EditText 中，如图 6.6 所示。

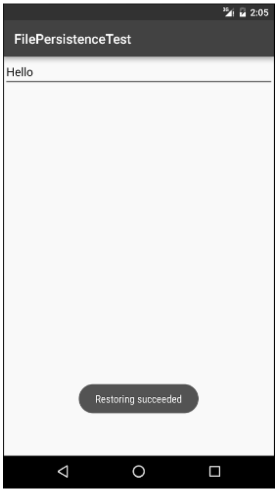


图 6.6 成功还原保存的内容

这样我们就已经把文件存储方面的知识学习完了，其实所用到的核心技术就是 `Context` 类中提供的 `openFileInput()`和 `openFileOutput()`方法，之后就是利用 Java 的各种流来进行读写操作。

不过正如我前面所说，文件存储的方式并不适合用于保存一些较为复杂的文本数据，因此，下面我们就来学习一下 Android 中另一种数据持久化的方式，它比文件存储更加简单易用，而且可以很方便地对某一指定的数据进行读写操作。

6.3 SharedPreferences 存储

不同于文件的存储方式，SharedPreferences 是使用键值对的方式来存储数据的。也就是说，当保存一条数据的时候，需要给这条数据提供一个对应的键，这样在读取数据的时候就可以通过这个键把相应的值取出来。而且 SharedPreferences 还支持多种不同的数据类型存储，如果存储的数据类型是整型，那么读取出来的数据也是整型的；如果存储的数据是一个字符串，那么读取出来的数据仍然是字符串。

这样你应该就能明显地感觉到，使用 SharedPreferences 来进行数据持久化要比使用文件方便很多，下面我们就来看一下它的具体用法吧。

6.3.1 将数据存储到 SharedPreferences 中

要想使用 SharedPreferences 来存储数据，首先需要获取到 `SharedPreferences` 对象。Android

中主要提供了 3 种方法用于得到 SharedPreferences 对象。

1. Context 类中的 getSharedPreferences()方法

此方法接收两个参数，第一个参数用于指定 SharedPreferences 文件的名称，如果指定的文件不存在则会创建一个，SharedPreferences 文件都是存放在/data/data/<package name>/shared_prefs/目录下的。第二个参数用于指定操作模式，目前只有 MODE_PRIVATE 这一种模式可选，它是默认的操作模式，和直接传入 0 效果是相同的，表示只有当前的应用程序才可以对这个 SharedPreferences 文件进行读写。其他几种操作模式均已被废弃，MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE 这两种模式是在 Android 4.2 版本中被废弃的，MODE_MULTI_PROCESS 模式是在 Android 6.0 版本中被废弃的。

2. Activity 类中的 getPreferences()方法

这个方法和 Context 中的 getSharedPreferences()方法很相似，不过它只接收一个操作模式参数，因为使用这个方法时会自动将当前活动的类名作为 SharedPreferences 的文件名。

3. PreferenceManager 类中的 getDefaultSharedPreferences()方法

这是一个静态方法，它接收一个 Context 参数，并自动使用当前应用程序的包名作为前缀来命名 SharedPreferences 文件。得到了 SharedPreferences 对象之后，就可以开始向 SharedPreferences 文件中存储数据了，主要可以分为 3 步实现。

(1) 调用 SharedPreferences 对象的 edit()方法来获取一个 SharedPreferences.Editor 对象。

(2) 向 SharedPreferences.Editor 对象中添加数据，比如添加一个布尔型数据就使用 putBoolean()方法，添加一个字符串则使用 putString()方法，以此类推。

(3) 调用 apply()方法将添加的数据提交，从而完成数据存储操作。

不知不觉中已经将理论知识介绍得挺多了，那我们就赶快通过一个例子来体验一下 SharedPreferences 存储的用法吧。新建一个 SharedPreferencesTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/save_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Save data"
        />

</LinearLayout>
```

这里我们不做任何复杂的功能，只是简单地放置了一个按钮，用于将一些数据存储到 SharedPreferences 文件当中。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button saveData = (Button) findViewById(R.id.save_data);
        saveData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences.Editor editor = getSharedPreferences("data",
                    MODE_PRIVATE).edit();
                editor.putString("name", "Tom");
                editor.putInt("age", 28);
                editor.putBoolean("married", false);
                editor.apply();
            }
        });
    }

}
```

可以看到，这里首先给按钮注册了一个点击事件，然后在点击事件中通过 getSharedPreferences()方法指定 SharedPreferences 的文件名为 data，并得到了 SharedPreferences.Editor 对象。接着向这个对象中添加了 3 条不同类型的数据，最后调用 apply()方法进行提交，从而完成了数据存储的操作。

很简单吧？现在就可以运行一下程序了，进入程序的主界面后，点击一下 Save data 按钮。这时的数据应该已经保存成功了，不过为了证实一下，我们还是要借助 File Explorer 来进行查看。打开 Android Device Monitor，并点击 File Explorer 标签页，然后进入到/data/data/com.example.sharedpreferencestest/shared_prefs/目录下，可以看到生成了一个 data.xml 文件，如图 6.7 所示。

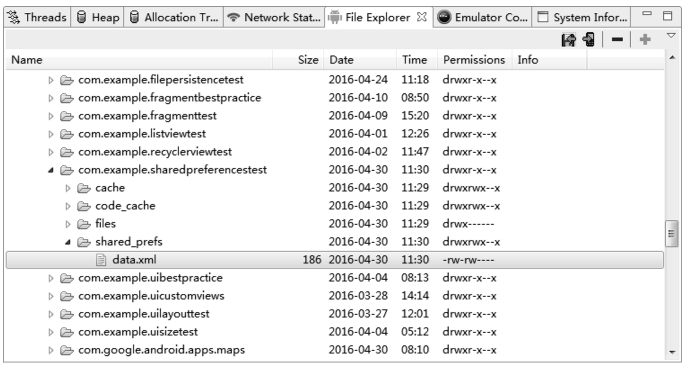


图 6.7 生成的 data.xml 文件

接下来，同样是点击导出按钮将这个文件导出到电脑上，并用记事本进行查看，里面的内容如图 6.8 所示。

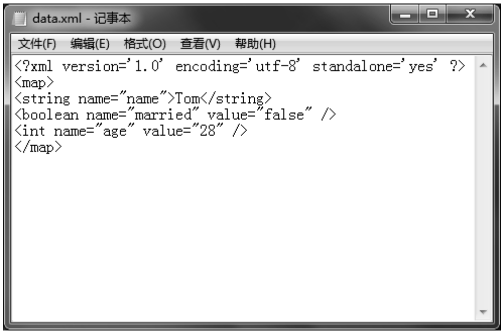


图 6.8　data.xml 文件中的内容

可以看到，我们刚刚在按钮的点击事件中添加的所有数据都已经成功保存下来了，并且 SharedPreferences 文件是使用 XML 格式来对数据进行管理的。

那么接下来我们自然要看一看，如何从 SharedPreferences 文件中去读取这些数据了。

6.3.2　从 SharedPreferences 中读取数据

你应该已经感觉到了，使用 SharedPreferences 来存储数据是非常简单的，不过下面还有更好的消息，其实从 SharedPreferences 文件中读取数据会更加地简单。SharedPreferences 对象中提供了一系列的 get 方法，用于对存储的数据进行读取，每种 get 方法都对应了 Shared-Preferences.Editor 中的一种 put 方法，比如读取一个布尔型数据就使用 getBoolean()方法，读取一个字符串就使用 getString()方法。这些 get 方法都接收两个参数，第一个参数是键，传入存储数据时使用的键就可以得到相应的值了；第二个参数是默认值，即表示当传入的键找不到对应的值时会以什么样的默认值进行返回。

我们还是通过例子来实际体验一下吧，仍然是在 SharedPreferencesTest 项目的基础上继续开发，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/save_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Save data"
        />
```

```
    <Button
        android:id="@+id/restore_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Restore data"
        />

</LinearLayout>
```

这里增加了一个还原数据的按钮，我们希望通过点击这个按钮来从 SharedPreferences 文件中读取数据。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        Button restoreData = (Button) findViewById(R.id.restore_data);
        restoreData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SharedPreferences pref = getSharedPreferences("data", MODE_PRIVATE);
                String name = pref.getString("name", "");
                int age = pref.getInt("age", 0);
                boolean married = pref.getBoolean("married", false);
                Log.d("MainActivity", "name is " + name);
                Log.d("MainActivity", "age is " + age);
                Log.d("MainActivity", "married is " + married);
            }
        });
    }

}
```

可以看到，我们在还原数据按钮的点击事件中首先通过 `getSharedPreferences()`方法得到了 `SharedPreferences` 对象，然后分别调用它的 `getString()`、`getInt()`和 `getBoolean()` 方法，去获取前面所存储的姓名、年龄和是否已婚，如果没有找到相应的值，就会使用方法中传入的默认值来代替，最后通过 Log 将这些值打印出来。

现在重新运行一下程序，并点击界面上的 Restore data 按钮，然后查看 logcat 中的打印信息，如图 6.9 所示。

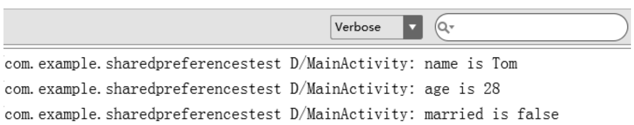


图 6.9　打印 data.xml 中存储的内容

所有之前存储的数据都成功读取出来了！通过这个例子，我们就把 SharedPreferences 存储

的知识也学习完了。相比之下，SharedPreferences 存储确实要比文本存储简单方便了许多，应用场景也多了不少，比如很多应用程序中的偏好设置功能其实都使用到了 SharedPreferences 技术。那么下面我们就来编写一个记住密码的功能，相信通过这个例子能够加深你对 SharedPreferences 的理解。

6.3.3　实现记住密码功能

既然是实现记住密码的功能，那么我们就不需要从头去写了，因为在上一章中的最佳实践部分已经编写过一个登录界面了，有可以重用的代码为什么不用呢？那就首先打开 Broadcast-BestPractice 项目，来编辑一下登录界面的布局。修改 activity_login.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <CheckBox
            android:id="@+id/remember_pass"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="18sp"
            android:text="Remember password" />
    </LinearLayout>

    <Button
        android:id="@+id/login"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:text="Login" />
</LinearLayout>
```

这里使用到了一个新控件 CheckBox。这是一个复选框控件，用户可以通过点击的方式来进行选中和取消，我们就使用这个控件来表示用户是否需要记住密码。

然后修改 LoginActivity 中的代码，如下所示：

```
public class LoginActivity extends BaseActivity {

    private SharedPreferences pref;

    private SharedPreferences.Editor editor;
```

```
private EditText accountEdit;

private EditText passwordEdit;

private Button login;

private CheckBox rememberPass;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);
    pref = PreferenceManager.getDefaultSharedPreferences(this);
    accountEdit = (EditText) findViewById(R.id.account);
    passwordEdit = (EditText) findViewById(R.id.password);
    rememberPass = (CheckBox) findViewById(R.id.remember_pass);
    login = (Button) findViewById(R.id.login);
    boolean isRemember = pref.getBoolean("remember_password", false);
    if (isRemember) {
        // 将账号和密码都设置到文本框中
        String account = pref.getString("account", "");
        String password = pref.getString("password", "");
        accountEdit.setText(account);
        passwordEdit.setText(password);
        rememberPass.setChecked(true);
    }
    login.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String account = accountEdit.getText().toString();
            String password = passwordEdit.getText().toString();
            // 如果账号是admin且密码是123456，就认为登录成功
            if (account.equals("admin") && password.equals("123456")) {
                editor = pref.edit();
                if (rememberPass.isChecked()) { // 检查复选框是否被选中
                    editor.putBoolean("remember_password", true);
                    editor.putString("account", account);
                    editor.putString("password", password);
                } else {
                    editor.clear();
                }
                editor.apply();
                Intent intent = new Intent(LoginActivity.this, MainActivity.
                    class);
                startActivity(intent);
                finish();
            } else {
                Toast.makeText(LoginActivity.this, "account or password is
                    invalid",
                    Toast.LENGTH_SHORT).show();
            }
        }
    });
```

```
    }
}
```

可以看到，这里首先在 onCreate()方法中获取到了 SharedPreferences 对象，然后调用它的 getBoolean()方法去获取 remember_password 这个键对应的值。一开始当然不存在对应的值了，所以会使用默认值 false，这样就什么都不会发生。接着在登录成功之后，会调用 CheckBox 的 isChecked()方法来检查复选框是否被选中，如果被选中了，则表示用户想要记住密码，这时将 remember_password 设置为 true，然后把 account 和 password 对应的值都存入到 SharedPreferences 文件当中并提交。如果没有被选中，就简单地调用一下 clear()方法，将 SharedPreferences 文件中的数据全部清除掉。

当用户选中了记住密码复选框，并成功登录一次之后，remember_password 键对应的值就是 true 了，这个时候如果再重新启动登录界面，就会从 SharedPreferences 文件中将保存的账号和密码都读取出来，并填充到文本输入框中，然后把记住密码复选框选中，这样就完成记住密码的功能了。

现在重新运行一下程序，可以看到界面上多出了一个记住密码复选框，如图 6.10 所示。

然后账号输入 admin，密码输入 123456，并选中记住密码复选框，点击登录，就会跳转到 MainActivity。接着在 MainActivity 中发出一条强制下线广播，会让程序重新回到登录界面，此时你会发现，账号密码都已经自动填充到界面上了，如图 6.11 所示。

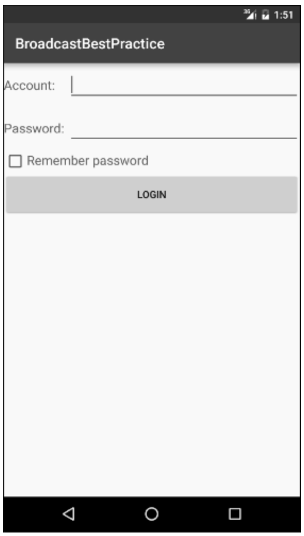


图 6.10　带记住密码复选框的登录界面



图 6.11　实现记住账号密码功能

这样我们就使用 SharedPreferences 技术将记住密码功能成功实现了，你是不是对 SharedPreferences 理解得更加深刻了呢？

不过需要注意，这里实现的记住密码功能仍然只是个简单的示例，并不能在实际的项目中直接使用。因为将密码以明文的形式存储在 SharedPreferences 文件中是非常不安全的，很容易就会被别人盗取，因此在正式的项目里还需要结合一定的加密算法来对密码进行保护才行。

好了，关于 SharedPreferences 的内容就讲到这里，接下来我们要学习一下本章的重头戏——Android 中的数据库技术。

6.4　SQLite 数据库存储

在刚开始接触 Android 的时候，我甚至都不敢相信，Android 系统竟然是内置了数据库的！好吧，是我太孤陋寡闻了。SQLite 是一款轻量级的关系型数据库，它的运算速度非常快，占用资源很少，通常只需要几百 KB 的内存就足够了，因而特别适合在移动设备上使用。SQLite 不仅支持标准的 SQL 语法，还遵循了数据库的 ACID 事务，所以只要你以前使用过其他的关系型数据库，就可以很快地上手 SQLite。而 SQLite 又比一般的数据库要简单得多，它甚至不用设置用户名和密码就可以使用。Android 正是把这个功能极为强大的数据库嵌入到了系统当中，使得本地持久化的功能有了一次质的飞跃。

前面我们所学的文件存储和SharedPreferences存储毕竟只适用于保存一些简单的数据和键值对，当需要存储大量复杂的关系型数据的时候，你就会发现以上两种存储方式很难应付得了。比如我们手机的短信程序中可能会有很多个会话，每个会话中又包含了很多条信息内容，并且大部分会话还可能各自对应了电话簿中的某个联系人。很难想象如何用文件或者 SharedPreferences 来存储这些数据量大、结构性复杂的数据吧？但是使用数据库就可以做得到。那么我们就赶快来看一看，Android 中的 SQLite 数据库到底是如何使用的。

6.4.1　创建数据库

Android 为了让我们能够更加方便地管理数据库，专门提供了一个 SQLiteOpenHelper 帮助类，借助这个类就可以非常简单地对数据库进行创建和升级。既然有好东西可以直接使用，那我们自然要尝试一下了，下面我就对 SQLiteOpenHelper 的基本用法进行介绍。

首先你要知道 SQLiteOpenHelper 是一个抽象类，这意味着如果我们想要使用它的话，就需要创建一个自己的帮助类去继承它。SQLiteOpenHelper 中有两个抽象方法，分别是 `onCreate()` 和 `onUpgrade()`，我们必须在自己的帮助类里面重写这两个方法，然后分别在这两个方法中去实现创建、升级数据库的逻辑。

SQLiteOpenHelper 中还有两个非常重要的实例方法：`getReadableDatabase()`和 `get-WritableDatabase()`。这两个方法都可以创建或打开一个现有的数据库（如果数据库已存在则直接打开，否则创建一个新的数据库），并返回一个可对数据库进行读写操作的对象。不同的是，当数据库不可写入的时候（如磁盘空间已满），`getReadableDatabase()`方法返回的对象将以只

读的方式去打开数据库，而 getWritableDatabase()方法则将出现异常。

SQLiteOpenHelper 中有两个构造方法可供重写，一般使用参数少一点的那个构造方法即可。这个构造方法中接收 4 个参数，第一个参数是 Context，这个没什么好说的，必须要有它才能对数据库进行操作。第二个参数是数据库名，创建数据库时使用的就是这里指定的名称。第三个参数允许我们在查询数据的时候返回一个自定义的 Cursor，一般都是传入 null。第四个参数表示当前数据库的版本号，可用于对数据库进行升级操作。构建出 SQLiteOpenHelper 的实例之后，再调用它的 getReadableDatabase()或 getWritableDatabase()方法就能够创建数据库了，数据库文件会存放在/data/data/<package name>/databases/目录下。此时，重写的 onCreate()方法也会得到执行，所以通常会在这里去处理一些创建表的逻辑。

接下来还是让我们通过例子的方式来更加直观地体会 SQLiteOpenHelper 的用法吧，首先新建一个 DatabaseTest 项目。

这里我们希望创建一个名为 BookStore.db 的数据库，然后在这个数据库中新建一张 Book 表，表中有 id（主键）、作者、价格、页数和书名等列。创建数据库表当然还是需要用建表语句的，这里也是要考验一下你的 SQL 基本功了，Book 表的建表语句如下所示：

```
create table Book (
    id integer primary key autoincrement,
    author text,
    price real,
    pages integer,
    name text)
```

只要你对 SQL 方面的知识稍微有一些了解，上面的建表语句对你来说应该都不难吧。SQLite 不像其他的数据库拥有众多繁杂的数据类型，它的数据类型很简单，integer 表示整型，real 表示浮点型，text 表示文本类型，blob 表示二进制类型。另外，上述建表语句中我们还使用了 primary key 将 id 列设为主键，并用 autoincrement 关键字表示 id 列是自增长的。

然后需要在代码中去执行这条 SQL 语句，才能完成创建表的操作。新建 MyDatabaseHelper 类继承自 SQLiteOpenHelper，代码如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table Book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text)";

    private Context mContext;

    public MyDatabaseHelper(Context context, String name,
                            SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
```

```
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
        Toast.makeText(mContext, "Create succeeded", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }

}
```

可以看到，我们把建表语句定义成了一个字符串常量，然后在 onCreate()方法中又调用了 SQLiteDatabase 的 execSQL()方法去执行这条建表语句，并弹出一个 Toast 提示创建成功，这样就可以保证在数据库创建完成的同时还能成功创建 Book 表。

现在修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <Button
        android:id="@+id/create_database"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Create database"
        />

</LinearLayout>
```

布局文件很简单，就是加入了一个按钮，用于创建数据库。最后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 1);
        Button createDatabase = (Button) findViewById(R.id.create_database);
        createDatabase.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                dbHelper.getWritableDatabase();
            }
```

```
        });
    }

}
```

这里我们在 `onCreate()`方法中构建了一个 `MyDatabaseHelper` 对象，并且通过构造函数的参数将数据库名指定为 BookStore.db，版本号指定为 1，然后在 Create database 按钮的点击事件里调用了 `getWritableDatabase()`方法。这样当第一次点击 Create database 按钮时，就会检测到当前程序中并没有 BookStore.db 这个数据库，于是会创建该数据库并调用 MyDatabaseHelper 中的 `onCreate()`方法，这样 Book 表也就得到了创建，然后会弹出一个 Toast 提示创建成功。再次点击 Create database 按钮时，会发现此时已经存在 BookStore.db 数据库了，因此不会再创建一次。

现在就可以运行一下代码了，在程序主界面点击 Create database 按钮，结果如图 6.12 所示。

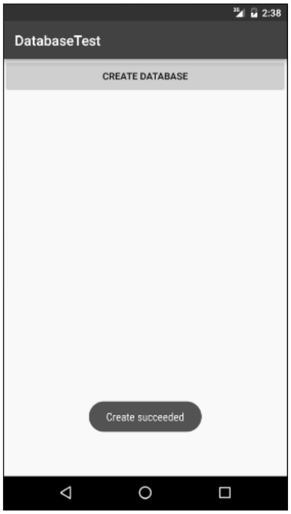


图 6.12　创建数据库成功

此时 BookStore.db 数据库和 Book 表应该都已经创建成功了，因为当你再次点击 Create database 按钮时，不会再有 Toast 弹出。可是又回到了之前的那个老问题，怎样才能证实它们的确创建成功了？如果还是使用 File Explorer，那么最多你只能看到 databases 目录下出现了一个 BookStore.db 文件，Book 表是无法通过 File Explorer 看到的。因此这次我们准备换一种查看方式，使用 adb shell 来对数据库和表的创建情况进行检查。

adb 是 Android SDK 中自带的一个调试工具，使用这个工具可以直接对连接在电脑上的手机或模拟器进行调试操作。它存放在 sdk 的 platform-tools 目录下，如果想要在命令行中使用这个工具，就需要先把它的路径配置到环境变量里。

如果你使用的是 Windows 系统，可以右击计算机→属性→高级系统设置→环境变量，然后在系统变量里找到 Path 并点击编辑，将 platform-tools 目录配置进去，如图 6.13 所示。



图 6.13　Windows 下配置环境变量

如果你使用的是 Linux 或 Mac 系统，可以在 home 路径下编辑.bash_文件，将 platform-tools 目录配置进去即可，如图 6.14 所示。

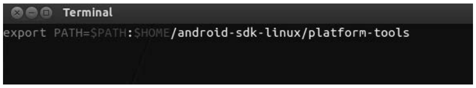


图 6.14　Linux 或 Mac 下配置环境变量

配置好了环境变量之后，就可以使用 adb 工具了。打开命令行界面，输入 `adb shell`，就会进入到设备的控制台，如图 6.15 所示。



图 6.15　进入设备的控制台

其中，#符号是超级管理员的意思，也就是说现在你可以访问模拟器中的一切数据。如果你的命令行上显示的是$符号，那么就表示你现在是普遍管理员，需输入 su 命令切换成超级管理员，才能执行下面的操作。

接下来使用 `cd` 命令进入到/data/data/com.example.databasetest/databases/目录下，并使用 `ls` 命令查看到该目录里的文件，如图 6.16 所示。

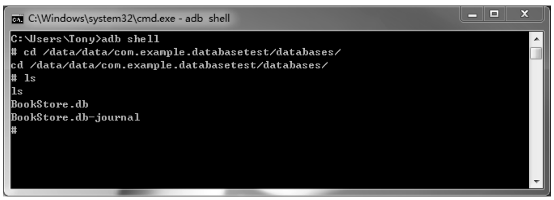


图 6.16　查看数据库文件

这个目录下出现了两个数据库文件，一个正是我们创建的 BookStore.db，而另一个 BookStore.db-journal 则是为了让数据库能够支持事务而产生的临时日志文件，通常情况下这个文件的大小都是 0 字节。

接下来我们就要借助 `sqlite` 命令来打开数据库了，只需要键入 sqlite3，后面加上数据库名即可，如图 6.17 所示。

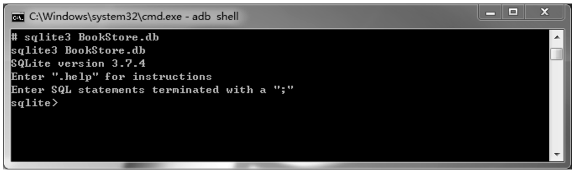


图 6.17　打开 BookStore.db 数据库

这时就已经打开了 BookStore.db 数据库，现在就可以对这个数据库中的表进行管理了。首先来看一下目前数据库中有哪些表，键入`.table` 命令，如图 6.18 所示。



图 6.18　查看表

可以看到，此时数据库中有两张表，`android_metadata` 表是每个数据库中都会自动生成的，不用管它，而另外一张 Book 表就是我们在 MyDatabaseHelper 中创建的了。这里还可以通过`.schema` 命令来查看它们的建表语句，如图 6.19 所示。



图 6.19　查看建表语句

由此证明，BookStore.db 数据库和 Book 表确实已经创建成功了。之后键入`.exit` 或`.quit` 命令可以退出数据库的编辑，再键入 `exit` 命令就可以退出设备控制台了。

6.4.2　升级数据库

如果你足够细心，一定会发现 MyDatabaseHelper 中还有一个空方法呢！没错，`onUpgrade()` 方法是用于对数据库进行升级的，它在整个数据库的管理工作当中起着非常重要的作用，可千万

不能忽视它哟。

目前 DatabaseTest 项目中已经有一张 Book 表用于存放书的各种详细数据，如果我们想再添加一张 Category 表用于记录图书的分类，该怎么做呢？

比如 Category 表中有 `id`（主键）、分类名和分类代码这几个列，那么建表语句就可以写成：

```
create table Category (
    id integer primary key autoincrement,
    category_name text,
    category_code integer)
```

接下来我们将这条建表语句添加到 MyDatabaseHelper 中，代码如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table Book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text)";

    public static final String CREATE_CATEGORY = "create table Category ("
            + "id integer primary key autoincrement, "
            + "category_name text, "
            + "category_code integer)";

    private Context mContext;

    public MyDatabaseHelper(Context context, String name,
                            SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
        db.execSQL(CREATE_CATEGORY);
        Toast.makeText(mContext, "Create succeeded", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }

}
```

看上去好像都挺对的吧？现在我们重新运行一下程序，并点击 Create database 按钮，咦？竟然没有弹出创建成功的提示。当然，你也可以通过 adb 工具到数据库中再去检查一下，这样你会更加地确认 Category 表没有创建成功！

其实没有创建成功的原因不难思考，因为此时 BookStore.db 数据库已经存在了，之后不管我们怎样点击 Create database 按钮，MyDatabaseHelper 中的 onCreate()方法都不会再次执行，因此新添加的表也就无法得到创建了。

解决这个问题的办法也相当简单，只需要先将程序卸载掉，然后重新运行，这时 BookStore.db 数据库已经不存在了，如果再点击 Create database 按钮，MyDatabaseHelper 中的 onCreate()方法就会执行，这时 Category 表就可以创建成功了。

不过，通过卸载程序的方式来新增一张表毫无疑问是很极端的做法，其实我们只需要巧妙地运用 SQLiteOpenHelper 的升级功能就可以很轻松地解决这个问题。修改 MyDatabaseHelper 中的代码，如下所示：

```
public class MyDatabaseHelper extends SQLiteOpenHelper {

    ...

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("drop table if exists Book");
        db.execSQL("drop table if exists Category");
        onCreate(db);
    }

}
```

可以看到，我们在 onUpgrade()方法中执行了两条 DROP 语句，如果发现数据库中已经存在 Book 表或 Category 表了，就将这两张表删除掉，然后再调用 onCreate()方法重新创建。这里先将已经存在的表删除掉，因为如果在创建表时发现这张表已经存在了，就会直接报错。

接下来的问题就是如何让 onUpgrade()方法能够执行了，还记得 SQLiteOpenHelper 的构造方法里接收的第四个参数吗？它表示当前数据库的版本号，之前我们传入的是 1，现在只要传入一个比 1 大的数，就可以让 onUpgrade()方法得到执行了。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        Button createDatabase = (Button) findViewById(R.id.create_database);
        createDatabase.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                dbHelper.getWritableDatabase();
            }
        });
```

```
    }
}
```

这里将数据库版本号指定为 2，表示我们对数据库进行升级了。现在重新运行程序，并点击 Create database 按钮，这时就会再次弹出创建成功的提示。为了验证一下 Category 表是不是已经创建成功了，我们在 `adb shell` 中打开 BookStore.db 数据库，然后键入`.table` 命令，结果如图 6.20 所示。



图 6.20　查看新增表

接着键入`.schema` 命令查看一下建表语句，结果如图 6.21 所示。

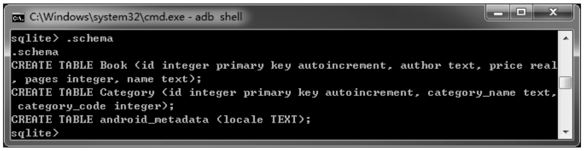


图 6.21　查看新增建表语句

由此可以看出，Category 表已经创建成功了，同时也说明我们的升级功能的确起到了作用。

6.4.3　添加数据

现在你已经掌握了创建和升级数据库的方法，接下来就该学习一下如何对表中的数据进行操作了。其实我们可以对数据进行的操作无非有 4 种，即 CRUD。其中 C 代表添加（Create），R 代表查询（Retrieve），U 代表更新（Update），D 代表删除（Delete）。每一种操作又各自对应了一种 SQL 命令，如果你比较熟悉 SQL 语言的话，一定会知道添加数据时使用 `insert`，查询数据时使用 `select`，更新数据时使用 `update`，删除数据时使用 `delete`。但是开发者的水平总会是参差不齐的，未必每一个人都能非常熟悉地使用 SQL 语言，因此 Android 也提供了一系列的辅助性方法，使得在 Android 中即使不去编写 SQL 语句，也能轻松完成所有的 CRUD 操作。

前面我们已经知道，调用 SQLiteOpenHelper 的 `getReadableDatabase()`或 `getWritableDatabase()`方法是可以用于创建和升级数据库的，不仅如此，这两个方法还都会返回一个 `SQLiteDatabase` 对象，借助这个对象就可以对数据进行 CRUD 操作了。

那么下面我们首先学习一下如何向数据库的表中添加数据吧。`SQLiteDatabase` 中提供了一个 `insert()`方法，这个方法就是专门用于添加数据的。它接收 3 个参数，第一个参数是表名，

我们希望向哪张表里添加数据，这里就传入该表的名字。第二个参数用于在未指定添加数据的情况下给某些可为空的列自动赋值 NULL，一般我们用不到这个功能，直接传入 null 即可。第三个参数是一个 ContentValues 对象，它提供了一系列的 put()方法重载，用于向 ContentValues 中添加数据，只需要将表中的每个列名以及相应的待添加数据传入即可。

介绍完了基本用法，接下来还是让我们通过例子的方式来亲身体验一下如何添加数据吧。修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    ...

    <Button
        android:id="@+id/add_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add data"
        />
</LinearLayout>
```

可以看到，我们在布局文件中又新增了一个按钮，稍后就会在这个按钮的点击事件里编写添加数据的逻辑。接着修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ...
        Button addData = (Button) findViewById(R.id.add_data);
        addData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                ContentValues values = new ContentValues();
                // 开始组装第一条数据
                values.put("name", "The Da Vinci Code");
                values.put("author", "Dan Brown");
                values.put("pages", 454);
                values.put("price", 16.96);
                db.insert("Book", null, values); // 插入第一条数据
                values.clear();
                // 开始组装第二条数据
                values.put("name", "The Lost Symbol");
                values.put("author", "Dan Brown");
```

```
                values.put("pages", 510);
                values.put("price", 19.95);
                db.insert("Book", null, values); // 插入第二条数据
            }
        });
    }

}
```

在添加数据按钮的点击事件里面，我们先获取到了 SQLiteDatabase 对象，然后使用 ContentValues 来对要添加的数据进行组装。如果你比较细心的话应该会发现，这里只对 Book 表里其中四列的数据进行了组装，id 那一列并没给它赋值。这是因为在前面创建表的时候，我们就将 id 列设置为自增长了，它的值会在入库的时候自动生成，所以不需要手动给它赋值了。接下来调用了 insert()方法将数据添加到表当中，注意这里我们实际上添加了两条数据，上述代码中使用 ContentValues 分别组装了两次不同的内容，并调用了两次 insert()方法。

好了，现在可以重新运行一下程序了，界面如图 6.22 所示。

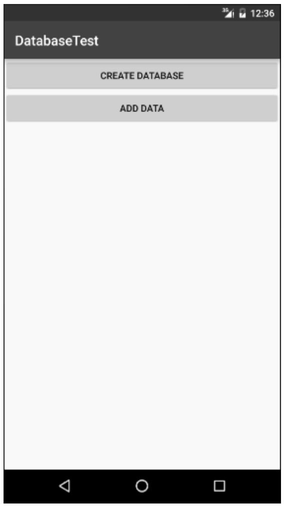


图 6.22　加入添加数据按钮

点击一下 Add data 按钮，此时两条数据应该都已经添加成功了，不过为了证实一下，我们还是打开 BookStore.db 数据库瞧一瞧。输入 SQL 查询语句 select * from Book;，结果如图 6.23 所示。



图 6.23　查看添加的数据

由此可以看出，我们刚刚组装的两条数据都已经准确无误地添加到 Book 表中了。

6.4.4　更新数据

学习完了如何向表中添加数据，接下来我们看看怎样才能修改表中已有的数据。SQLite-Database 中也提供了一个非常好用的 update()方法，用于对数据进行更新，这个方法接收 4 个参数，第一个参数和 insert()方法一样，也是表名，在这里指定去更新哪张表里的数据。第二个参数是 ContentValues 对象，要把更新数据在这里组装进去。第三、第四个参数用于约束更新某一行或某几行中的数据，不指定的话默认就是更新所有行。

那么接下来我们仍然是在 DatabaseTest 项目的基础上修改，看一下更新数据的具体用法。比如说刚才添加到数据库里的第一本书，由于过了畅销季，卖得不是很火了，现在需要通过降低价格的方式来吸引更多的顾客，我们应该怎么操作呢？首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    ...

    <Button
        android:id="@+id/update_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Update data"
        />
</LinearLayout>
```

布局文件中的代码已经非常简单了，就是添加了一个用于更新数据的按钮。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ...
        Button updateData = (Button) findViewById(R.id.update_data);
        updateData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                ContentValues values = new ContentValues();
                values.put("price", 10.99);
```

```
                db.update("Book", values, "name = ?", new String[] { "The Da Vinci
                    Code" });
            }
        });
    }

}
```

这里在更新数据按钮的点击事件里面构建了一个 ContentValues 对象，并且只给它指定了一组数据，说明我们只是想把价格这一列的数据更新成 10.99。然后调用了 SQLiteDatabase 的 update()方法去执行具体的更新操作，可以看到，这里使用了第三、第四个参数来指定具体更新哪几行。第三个参数对应的是 SQL 语句的 where 部分，表示更新所有 name 等于?的行，而?是一个占位符，可以通过第四个参数提供的一个字符串数组为第三个参数中的每个占位符指定相应的内容。因此上述代码想表达的意图是将名字是 The Da Vinci Code 的这本书的价格改成 10.99。

现在重新运行一下程序，界面如图 6.24 所示。

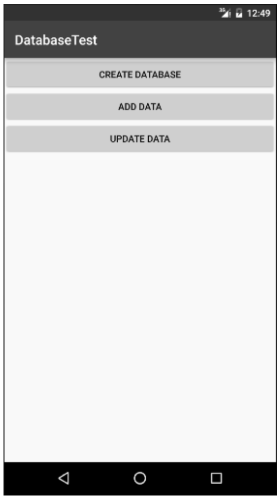


图 6.24 加入更新数据按钮

点击一下 Update data 按钮后，再次输入查询语句查看表中的数据情况，结果如图 6.25 所示。



图 6.25 查看更新后的数据

可以看到，The Da Vinci Code 这本书的价格已经被成功改为 10.99 了。

6.4.5　删除数据

怎么样？添加和更新数据的功能都还挺简单的吧，代码也不多，理解起来又容易，那么我们要马不停蹄地开始学习下一种操作了，即从表中删除数据。

删除数据对你来说应该就更简单了，因为它所需要用到的知识点你全部已经学过了。SQLiteDatabase 中提供了一个 delete()方法，专门用于删除数据，这个方法接收 3 个参数，第一个参数仍然是表名，这个已经没什么好说的了，第二、第三个参数又是用于约束删除某一行或某几行的数据，不指定的话默认就是删除所有行。

是不是理解起来很轻松了？那我们就继续动手实践吧，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    ...

    <Button
        android:id="@+id/delete_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Delete data"
        />
</LinearLayout>
```

仍然是在布局文件中添加了一个按钮，用于删除数据。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ...
        Button deleteButton = (Button) findViewById(R.id.delete_data);
        deleteButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                db.delete("Book", "pages > ?", new String[] { "500" });
            }
        });
```

```
    }
}
```

可以看到，我们在删除按钮的点击事件里指明去删除 Book 表中的数据，并且通过第二、第三个参数来指定仅删除那些页数超过 500 页的书。当然这个需求很奇怪，这里也仅仅是为了做个测试。你可以先查看一下当前 Book 表里的数据，其中 The Lost Symbol 这本书的页数超过了 500 页，也就是说当我们点击删除按钮时，这条记录应该会被删除掉。

现在重新运行一下程序，界面如图 6.26 所示。

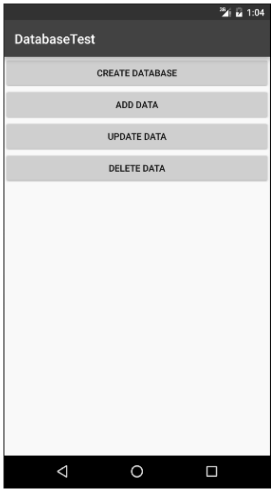


图 6.26　加入删除数据按钮

点击一下 Delete data 按钮后，再次输入查询语句查看表中的数据情况，结果如图 6.27 所示。



图 6.27　查看删除后的数据

6.4.6　查询数据

终于到了最后一种操作了，掌握了查询数据的方法之后，你就将数据库的 CRUD 操作全部学完了。不过千万不要因此而放松，因为查询数据是 CRUD 中最复杂的一种操作。

我们都知道 SQL 的全称是 Structured Query Language，翻译成中文就是结构化查询语言。它的大部分功能都体现在“查”这个字上的，而“增删改”只是其中的一小部分功能。由于 SQL 查

询涉及的内容实在是太多了，因此在这里我不准备对它展开来讲解，而是只会介绍 Android 上的查询功能。如果你对 SQL 语言非常感兴趣，可以找一本专门介绍 SQL 的书进行学习。

相信你已经猜到了，SQLiteDatabase 中还提供了一个 query()方法用于对数据进行查询。这个方法的参数非常复杂，最短的一个方法重载也需要传入 7 个参数。那我们就先来看一下这 7 个参数各自的含义吧。第一个参数不用说，当然还是表名，表示我们希望从哪张表中查询数据。第二个参数用于指定去查询哪几列，如果不指定则默认查询所有列。第三、第四个参数用于约束查询某一行或某几行的数据，不指定则默认查询所有行的数据。第五个参数用于指定需要去 group by 的列，不指定则表示不对查询结果进行 group by 操作。第六个参数用于对 group by 之后的数据进行进一步的过滤，不指定则表示不进行过滤。第七个参数用于指定查询结果的排序方式，不指定则表示使用默认的排序方式。更多详细的内容可以参考下表。其他几个 query()方法的重载其实也大同小异，你可以自己去研究一下，这里就不再进行介绍了。

query()方法参数	对应SQL部分	描　　述
table	from table_name	指定查询的表名
columns	select column1, column2	指定查询的列名
selection	where column = value	指定where的约束条件
selectionArgs	-	为where中的占位符提供具体的值
groupBy	group by column	指定需要group by的列
having	having column = value	对group by后的结果进一步约束
orderBy	order by column1, column2	指定查询结果的排序方式

虽然 query()方法的参数非常多，但是不要对它产生畏惧，因为我们不必为每条查询语句都指定所有的参数，多数情况下只需要传入少数几个参数就可以完成查询操作了。调用 query()方法后会返回一个 Cursor 对象，查询到的所有数据都将从这个对象中取出。

下面还是让我们通过例子的方式来体验一下查询数据的具体用法，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    ...

    <Button
        android:id="@+id/query_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Query data"
        />
</LinearLayout>
```

这个已经没什么好说的了，添加了一个按钮用于查询数据。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private MyDatabaseHelper dbHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new MyDatabaseHelper(this, "BookStore.db", null, 2);
        ...
        Button queryButton = (Button) findViewById(R.id.query_data);
        queryButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SQLiteDatabase db = dbHelper.getWritableDatabase();
                // 查询Book表中所有的数据
                Cursor cursor = db.query("Book", null, null, null, null, null, null);
                if (cursor.moveToFirst()) {
                    do {
                        // 遍历Cursor对象，取出数据并打印
                        String name = cursor.getString(cursor.getColumnIndex
                            ("name"));
                        String author = cursor.getString(cursor.getColumnIndex
                            ("author"));
                        int pages = cursor.getInt(cursor.getColumnIndex("pages"));
                        double price = cursor.getDouble(cursor.getColumnIndex
                            ("price"));
                        Log.d("MainActivity", "book name is " + name);
                        Log.d("MainActivity", "book author is " + author);
                        Log.d("MainActivity", "book pages is " + pages);
                        Log.d("MainActivity", "book price is " + price);
                    } while (cursor.moveToNext());
                }
                cursor.close();
            }
        });
    }

}
```

可以看到，我们首先在查询按钮的点击事件里面调用了 SQLiteDatabase 的 `query()`方法去查询数据。这里的 `query()`方法非常简单，只是使用了第一个参数指明去查询 Book 表，后面的参数全部为 `null`。这就表示希望查询这张表中的所有数据，虽然这张表中目前只剩下一条数据了。查询完之后就得到了一个 `Cursor` 对象，接着我们调用它的 `moveToFirst()`方法将数据的指针移动到第一行的位置，然后进入了一个循环当中，去遍历查询到的每一行数据。在这个循环中可以通过 `Cursor` 的 `getColumnIndex()`方法获取到某一列在表中对应的位置索引，然后将这个索引传入到相应的取值方法中，就可以得到从数据库中读取到的数据了。接着我们使用 Log 的方式将

取出的数据打印出来，借此来检查一下读取工作有没有成功完成。最后别忘了调用 `close()`方法来关闭 `Cursor`。

好了，现在再次重新运行程序，界面如图 6.28 所示。

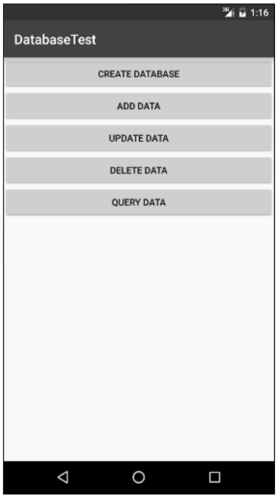


图 6.28　加入查询数据按钮

点击一下 Query data 按钮后，查看 logcat 的打印内容，结果如图 6.29 所示。

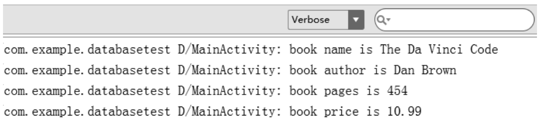


图 6.29　打印查询到的数据

可以看到，这里已经将 Book 表中唯一的一条数据成功地读取出来了。

当然这个例子只是对查询数据的用法进行了最简单的示范，在真正的项目中你可能会遇到比这要复杂得多的查询功能，更多高级的用法还需要你自己去慢慢摸索，毕竟 `query()`方法中还有那么多的参数我们都还没用到呢。

6.4.7　使用 SQL 操作数据库

虽然 Android 已经给我们提供了很多非常方便的 API 用于操作数据库，不过总会有一些人不习惯去使用这些辅助性的方法，而是更加青睐于直接使用 SQL 来操作数据库。这种人一般都属于 SQL 大牛，如果你也是其中之一的话，那么恭喜，Android 充分考虑到了你们的编程习惯，同样提供了一系列的方法，使得可以直接通过 SQL 来操作数据库。

下面我就来简略演示一下，如何直接使用 SQL 来完成前面几小节中学过的 CRUD 操作。

- 添加数据的方法如下：

```
db.execSQL("insert into Book (name, author, pages, price) values(?, ?, ?, ?)",
            new String[] { "The Da Vinci Code", "Dan Brown", "454", "16.96" });
db.execSQL("insert into Book (name, author, pages, price) values(?, ?, ?, ?)",
            new String[] { "The Lost Symbol", "Dan Brown", "510", "19.95" });
```

- 更新数据的方法如下：

```
db.execSQL("update Book set price = ? where name = ?", new String[] { "10.99",
"The Da Vinci Code" });
```

- 删除数据的方法如下：

```
db.execSQL("delete from Book where pages > ?", new String[] { "500" });
```

- 查询数据的方法如下：

```
db.rawQuery("select * from Book", null);
```

可以看到，除了查询数据的时候调用的是 SQLiteDatabase 的 `rawQuery()`方法，其他的操作都是调用的 `execSQL()`方法。以上演示的几种方式，执行结果会和前面几小节中我们学习的 CRUD 操作的结果完全相同，选择使用哪一种方式就看你个人的喜好了。

6.5　使用 LitePal 操作数据库

上一节中我们学习了使用 SQLiteDatabase 来操作 SQLite 数据库的方法，你觉得好用吗？每个人的回答可能会不一样。但我相信，等学完了本节的内容之后，你将再也不想去碰 SQLiteDatabase 了。到底是什么东西这么神奇？新建一个 LitePalTest 项目，然后开始我们本节的学习之旅吧。

6.5.1　LitePal 简介

如今，Android 的学习环境比起我当年学习的时候已经好太多了。当时国内做 Android 的人并不多，各种学习资料也比较欠缺，一个项目中几乎所有的功能都要完全靠自己从头来实现，开发效率之低下可想而知。

而现在开源的热潮让所有 Android 开发者都大大受益，GitHub 上面有成百上千的优秀 Android 开源项目，很多之前我们要写很久才能实现的功能，使用开源库可能短短几分钟就能实现了。除此之外，公司里的代码非常强调稳定性，而我们自己写出的代码往往越复杂就越容易出问题。相反，开源项目的代码都是经过时间验证的，通常比我们自己的代码要稳定得多。因此，现在有很多公司为了追求开发效率以及项目稳定性，都会选择使用开源库。

本书中我们将会学习多个开源库的使用方法，而现在你将正式开始接触第一个开源库——LitePal。

LitePal 是一款开源的 Android 数据库框架，它采用了对象关系映射（ORM）的模式，并将我们平时开发最常用到的一些数据库功能进行了封装，使得不用编写一行 SQL 语句就可以完成各种建表和增删改查的操作。LitePal 的项目主页上也有详细的使用文档，地址是：https://github.com/LitePalFramework/LitePal。

6.5.2　配置 LitePal

那么怎样才能在项目中使用开源库呢？过去的方式比较复杂，通常需要下载开源库的 Jar 包或者源码，然后再集成到我们的项目当中。而现在就简单得多了，大多数的开源项目都会将版本提交到 jcenter 上，我们只需要在 app/build.gradle 文件中声明该开源库的引用就可以了。

因此，要使用 LitePal 的第一步，就是编辑 app/build.gradle 文件，在 dependencies 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:23.2.0'
    testCompile 'junit:junit:4.12'
    compile 'org.litepal.android:core:1.4.1'
}
```

添加的这一行声明中，前面部分是固定的，最后的 1.4.1 是版本号的意思，最新的版本号可以到 LitePal 的项目主页上去查看。

这样我们就把 LitePal 成功引入到当前项目中了，接下来需要配置 litepal.xml 文件。右击 app/src/main 目录→New→Directory，创建一个 assets 目录，然后在 assets 目录下再新建一个 litepal.xml 文件，接着编辑 litepal.xml 文件中的内容，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<litepal>
    <dbname value="BookStore" ></dbname>

    <version value="1" ></version>

    <list>
    </list>
</litepal>
```

其中，`<dbname>`标签用于指定数据库名，`<version>`标签用于指定数据库版本号，`<list>`标签用于指定所有的映射模型，我们稍后就会用到。

最后还需要再配置一下 LitePalApplication，修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.litepaltest">
    <application
        android:name="org.litepal.LitePalApplication"
        android:allowBackup="true"
```

```
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
    </application>
</manifest>
```

这里我们将项目的 application 配置为 org.litepal.LitePalApplication，这样才能让 LitePal 的所有功能都可以正常工作。关于 application 的作用，我们之前并没有进行过详细的讲解，现在你只需要知道必须这么写就行了，我们将会在第 13 章中学习 application 的更多内容。

现在 LitePal 的配置工作已经全部结束了，下面我们开始正式使用它吧。

6.5.3 创建和升级数据库

我们之前创建数据库是通过自定义一个类继承自 SQLiteOpenHelper，然后在 onCreate()方法中编写建表语句来实现的，而使用 LitePal 就不用再这么麻烦了。本节中我们会使用 LitePal 来逐一完成上一节中所学的所有功能，以此来对比它们之间的差距，那么为了方便测试，我们先将 activity_main.xml 布局文件从 DatabaseTest 项目复制到 LitePalTest 项目中来。

刚才在介绍的时候已经说过，LitePal 采取的是对象关系映射（ORM）的模式，那么什么是对象关系映射呢？简单点说，我们使用的编程语言是面向对象语言，而使用的数据库则是关系型数据库，那么将面向对象的语言和面向关系的数据库之间建立一种映射关系，这就是对象关系映射了。

不过你可千万不要小看对象关系映射模式，它赋予了我们一个强大的功能，就是可以用面向对象的思维来操作数据库，而不用再和 SQL 语句打交道了，不信的话我们现在就来体验一下。比如在 6.4.1 小节中，为了创建一张 Book 表，需要先分析表中应该包含哪些列，然后再编写出一条建表语句，最后在自定义的 SQLiteOpenHelper 中去执行这条建表语句。但是使用 LitePal，你就可以用面向对象的思维来实现同样的功能了，定义一个 Book 类，代码如下所示：

```
public class Book {

    private int id;

    private String author;

    private double price;

    private int pages;

    private String name;

    public int getId() {
        return id;
    }
```

```
    public void setId(int id) {
        this.id = id;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public int getPages() {
        return pages;
    }

    public void setPages(int pages) {
        this.pages = pages;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

这是一个典型的 Java bean，在 Book 类中我们定义了 id、author、price、pages、name 这几个字段，并生成了相应的 getter 和 setter 方法。[①]相应你已经能猜到了，Book 类就会对应数据库中的 Book 表，而类中的每一个字段分别对应了表中的每一个列，这就是对象关系映射最直观的体验，现在你能够理解得更加清楚了吧。

接下来我们还需要将 Book 类添加到映射模型列表当中，修改 litepal.xml 中的代码，如下所示：

```
<litepal>
    <dbname value="BookStore" ></dbname>
```

① 生成 getter 和 setter 方法的快捷方式是，先将类中的字段定义好，然后按下 Alt + Insert 键（Mac 系统是 command + N），在弹出菜单中选择 Getter and Setter，接着使用 Shift 键将所有字段都选中，最后点击 OK。

```
    <version value="1" ></version>

    <list>
        <mapping class="com.example.litepaltest.Book"></mapping>
    </list>
</litepal>
```

这里使用`<mapping>`标签来声明我们要配置的映射模型类，注意一定要使用完整的类名。不管有多少模型类需要映射，都使用同样的方式配置在`<list>`标签下即可。

没错，这样就已经把所有工作都完成了，现在只要进行任意一次数据库的操作，BookStore.db 数据库应该就会自动创建出来。那么我们修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button createDatabase = (Button) findViewById(R.id.create_database);
        createDatabase.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                LitePal.getDatabase();
            }
        });
    }

}
```

其中，调用 `LitePal.getDatabase()`方法就是一次最简单的数据库操作，只要点击一下按钮，数据库就会自动创建完成了。运行一下程序，然后点击 Create database 按钮，接着通过 `adb shell` 查看一下数据库创建情况，如图 6.30 所示。

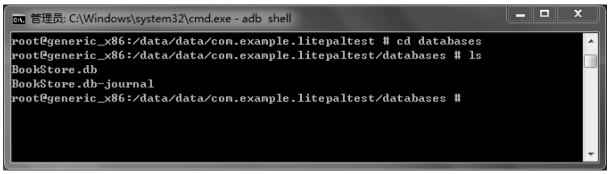


图 6.30　查看数据库文件

非常棒！数据库文件已经创建成功了。接下来我们使用 `sqlite3` 命令打开 BookStore.db 文件，然后再使用`.schema` 命令来查看建表语句，如图 6.31 所示。

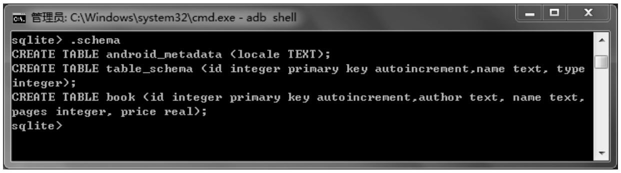


图 6.31　查看建表语句

可以看到，这里有 3 张表的建表语句，其中 android_metadata 表仍然不用管，table_schema 表是 LitePal 内部使用的，我们也可以直接忽视，book 表就是根据我们定义的 Book 类以及类中的字段来自动生成的了。

怎么样，是不是很神奇？但不用太吃惊，因为更加神奇的还在后面呢。6.4.2 节中我们体验了使用 SQLiteOpenHelper 来升级数据库的方式，虽说功能是实现了，但你有没有发现一个问题，就是升级数据库的时候我们需要先把之前的表 drop 掉，然后再重新创建才行。这其实是一个非常严重的问题，因为这样会造成数据丢失，每当升级一次数据库，之前表中的数据就全没了。

当然如果你是非常有经验的程序员，也可以通过复杂的逻辑控制来避免这种情况，但是维护成本很高。而有了 LitePal，这些就都不再是问题了，使用 LitePal 来升级数据库非常非常简单，你完全不用思考任何的逻辑，只需要改你想改的任何内容，然后将版本号加 1 就行了。

比如我们想要向 Book 表中添加一个 press（出版社）列，直接修改 Book 类中的代码，添加一个 press 字段即可，如下所示：

```
public class Book {

    ...

    private String press;

    ...

    public String getPress() {
        return press;
    }

    public void setPress(String press) {
        this.press = press;
    }

}
```

与此同时，我们还想再添加一张 Category 表，那么只需要新建一个 Category 类就可以了，代码如下所示：

```
public class Category {

    private int id;
```

```
    private String categoryName;

    private int categoryCode;

    public void setId(int id) {
        this.id = id;
    }

    public void setCategoryName(String categoryName) {
        this.categoryName = categoryName;
    }

    public void setCategoryCode(int categoryCode) {
        this.categoryCode = categoryCode;
    }

}
```

改完了所有我们想改的东西，只需要记得将版本号加 1 就行了。当然由于这里还添加了一个新的模型类，因此也需要将它添加到映射模型列表中。修改 litepal.xml 中的代码，如下所示：

```
<litepal>
    <dbname value="BookStore" ></dbname>

    <version value="2" ></version>

    <list>
        <mapping class="com.example.litepaltest.Book"></mapping>
        <mapping class="com.example.litepaltest.Category"></mapping>
    </list>
</litepal>
```

现在重新运行一下程序，然后点击 Create database 按钮，再查看一下最新的建表语句，结果如图 6.32 所示。

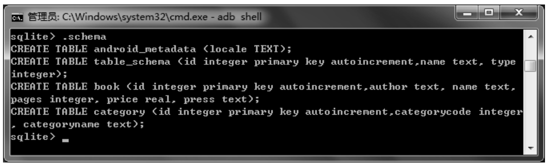


图 6.32　升级数据库后的建表语句

可以看到，book 表中新增了一个 press 列，category 表也创建成功了，当然 LitePal 还自动帮我们做了一项非常重要的工作，就是保留之前表中的所有数据，这样就再也不用担心数据丢失的问题了。

6.5.4　使用 LitePal 添加数据

体验了使用 LitePal 来创建和升级数据库，是不是感觉已经有一些小震撼了呢？不过 LitePal 所提供的强大功能还远不止于此，接下来我们就学习一下如何使用它来向数据库的表中添加数据吧。

首先回顾一下之前添加数据的方法，我们需要创建出一个 ContentValues 对象，然后将所有要添加的数据 put 到这个 ContentValues 对象当中，最后再调用 SQLiteDatabase 的 insert() 方法将数据添加到数据库表当中。

而使用 LitePal 来添加数据，这些操作可以简单到让你惊叹！我们只需要创建出模型类的实例，再将所有要存储的数据设置好，最后调用一下 save()方法就可以了。

下面开始来动手实现，观察现有的模型类，你会发现它们都是没有继承结构的。没错，因为 LitePal 进行表管理操作时不需要模型类有任何的继承结构，但是进行 CRUD 操作时就不行了，必须要继承自 DataSupport 类才行，因此这里我们需要先把继承结构给加上。修改 Book 类中的代码，如下所示：

```
public class Book extends DataSupport {
    ...
}
```

接着我们开始向 Book 表中添加数据，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        Button addData = (Button) findViewById(R.id.add_data);
        addData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Book book = new Book();
                book.setName("The Da Vinci Code");
                book.setAuthor("Dan Brown");
                book.setPages(454);
                book.setPrice(16.96);
                book.setPress("Unknow");
                book.save();
            }
        });
    }

}
```

这段代码非常神奇，我们来仔细阅读一下。在添加数据按钮的点击事件里面，首先是创建出

了一个 Book 的实例，然后调用 Book 类中的各种 set 方法对数据进行设置，最后再调用 book.save()方法就能完成数据添加操作了。那么这个 save()方法是从哪儿来的呢？当然是从 DataSupport 类中继承而来的了。除了 save()方法之外，DataSupport 类还给我们提供了丰富的 CRUD 方法，这些我们在后面都会学到。

现在重新运行程序，点击一下 Add data 按钮，此时数据应该已经添加成功了，我们打开 BookStore.db 数据库瞧一瞧。输入 SQL 查询语句 select * from Book，结果如图 6.33 所示。

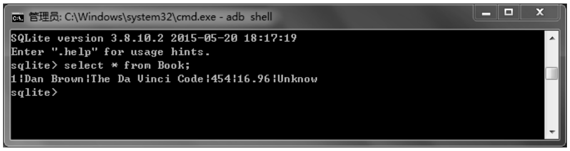


图 6.33　查看添加的数据

可以看到，作者、书名、页数、价格、出版社，这些数据全部精确无误地添加成功了。

6.5.5　使用 LitePal 更新数据

学习完了如何使用 LitePal 添加数据，接下来我们看看怎样使用 LitePal 更新数据。更新数据要比添加数据稍微复杂一点，因为它的 API 接口比较多，这里我们只介绍最常用的几种更新方式。

首先，最简单的一种更新方式就是对已存储的对象重新设值，然后重新调用 save()方法即可。那么这里我们就要了解一个概念，什么是已存储的对象？

对于 LitePal 来说，对象是否已存储就是根据调用 model.isSaved()方法的结果来判断的，返回 true 就表示已存储，返回 false 就表示未存储。那么接下来的问题就是，什么情况下会返回 true，什么情况下会返回 false 呢？

实际上只有在两种情况下 model.isSaved()方法才会返回 true，一种情况是已经调用过 model.save()方法去添加数据了，此时 model 会被认为是已存储的对象。另一种情况是 model 对象是通过 LitePal 提供的查询 API 查出来的，由于是从数据库中查到的对象，因此也会被认为是已存储的对象。

由于查询 API 我们暂时还没学到，因此只能先通过第一种情况来进行验证。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        Button updateData = (Button) findViewById(R.id.update_data);
        updateData.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
                public void onClick(View v) {
                    Book book = new Book();
                    book.setName("The Lost Symbol");
                    book.setAuthor("Dan Brown");
                    book.setPages(510);
                    book.setPrice(19.95);
                    book.setPress("Unknow");
                    book.save();
                    book.setPrice(10.99);
                    book.save();
                }
            });
        }

    }
```

在更新数据按钮的点击事件里面，我们先是通过上一小节中学习的知识添加了一条 Book 数据，然后调用 `setPrice()`方法将这本书的价格进行了修改，之后再次调用了 `save()`方法。此时 LitePal 会发现当前的 Book 对象是已存储的，因此不会再向数据库中去添加一条新数据，而是会直接更新当前的数据。

现在重新运行一下程序，然后点击 Update data 按钮，我们再次输入查询语句查看表中的数据情况，结果如图 6.34 所示。

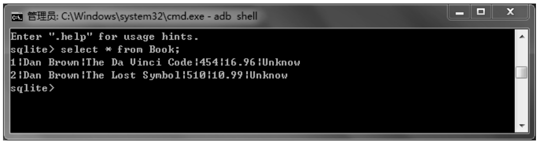


图 6.34　查看更新后的数据

可以看到，Book 表中新增了一条书的数据，但这本书的价格并不是一开始设置的 19.95，而是 10.99，说明我们的更新操作确实生效了。

但是这种更新方式只能对已存储的对象进行操作，限制性比较大，接下来我们学习另外一种更加灵巧的更新方式。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        Button updateData = (Button) findViewById(R.id.update_data);
        updateData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Book book = new Book();
                book.setPrice(14.95);
```

```
                    book.setPress("Anchor");
                    book.updateAll("name = ? and author = ?", "The Lost Symbol", "Dan
                        Brown");
                }
            });
        }

    }
```

可以看到，这里我们首先 new 出了一个 Book 的实例，然后直接调用 setPrice()和 setPress()方法来设置要更新的数据，最后再调用 updateAll()方法去执行更新操作。注意 updateAll()方法中可以指定一个条件约束，和 SQLiteDatabase 中 update()方法的 where 参数部分有点类似，但更加简洁，如果不指定条件语句的话，就表示更新所有数据。这里我们指定将所有书名是 The Lost Symbol 并且作者是 Dan Brown 的书价格更新为 14.95，出版社更新为 Anchor。

现在重新运行程序并点击 Update data 按钮，我们再次查询一下表中的数据情况，结果如图 6.35 所示。

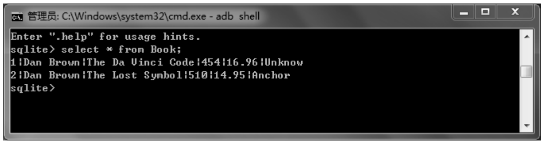


图 6.35 再次查看更新后的数据

意料之中，第二本书的价格被更新成了 14.95，出版社被更新成了 Anchor。怎么样？LitePal 的更新 API 是不是明显比 SQLiteDatabase 的 update()方法要好用多了？

不过，在使用 updateAll()方法时，还有一个非常重要的知识点是你需要知晓的，就是当你想把一个字段的值更新成默认值时，是不可以使用上面的方式来 set 数据的。我们都知道，在 Java 中任何一种数据类型的字段都会有默认值，例如 int 类型的默认值是 0，boolean 类型的默认值是 false，String 类型的默认值是 null。那么当 new 出一个 Book 对象时，其实所有字段都已经被初识化成默认值了，比如说 pages 字段的值就是 0。因此，如果我们想把数据库表中的 pages 列更新成 0，直接调用 book.setPages(0)是不可以的，因为即使不调用这行代码，pages 字段本身也是 0，LitePal 此时是不会对这个列进行更新的。对于所有想要将为数据更新成默认值的操作，LitePal 统一提供了一个 setToDefault()方法，然后传入相应的列名就可以实现了。比如我们可以这样写：

```
Book book = new Book();
book.setToDefault("pages");
book.updateAll();
```

这段代码的意思是，将所有书的页数都更新为 0，因为 updateAll()方法中没有指定约束条件，因此更新操作对所有数据都生效了。

6.5.6　使用 LitePal 删除数据

使用 LitePal 删除数据的方式主要有两种，第一种比较简单，就是直接调用已存储对象的 delete()方法就可以了，对于已存储对象的概念，我们在上一小节中已经学习过了。也就是说，调用过 save()方法的对象，或者是通过 LitePal 提供的查询 API 查出来的对象，都是可以直接使用 delete()方法来删除数据的。这种方式比较简单，我们就不进行代码演示了，下面直接来看另外一种删除数据的方式。

修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        Button deleteButton = (Button) findViewById(R.id.delete_data);
        deleteButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                DataSupport.deleteAll(Book.class, "price < ?", "15");
            }
        });
    }

}
```

这里调用了 DataSupport.deleteAll()方法来删除数据，其中 deleteAll()方法的第一个参数用于指定删除哪张表中的数据，Book.class 就意味着删除 Book 表中的数据，后面的参数用于指定约束条件，应该不难理解。那么这行代码的意思就是，删除 Book 表中价格低于 15 的书，正好目前 Book 表中有两本书，一本价格是 16.96，一本价格是 14.95，刚好可以看出效果。

现在重新运行程序，并点击一下 Delete data 按钮，然后查询表中的数据情况，如图 6.36 所示。

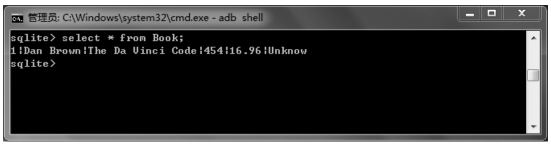


图 6.36　查看删除后的数据

可以看到，价格低于 15 的那本书已经被删除掉了。

另外，deleteAll()方法如果不指定约束条件，就意味着你要删除表中的所有数据，这一点和 updateAll()方法是比较相似的。

6.5.7 使用 LitePal 查询数据

终于又到了最复杂的查询数据部分了，不过这个“最复杂”只是相对于过去而言，因为使用 LitePal 来查询数据一点都不复杂。我一直都认为 LitePal 在查询 API 方面的设计极为人性化，想想之前我们所使用的 `query()`方法，冗长的参数列表让人看得头疼，即使多数参数都是用不到的，也不得不传入 `null`，如下所示：

```
Cursor cursor = db.query("Book", null, null, null, null, null, null);
```

像这样的代码恐怕是没人会喜欢的。为此 LitePal 在查询 API 方面做了非常多的优化，基本上可以满足绝大多数场景的查询需求，并且代码十分整洁，下面我们就来一起学习一下。

首先分析一下上述代码，`query()`方法中使用了第一个参数指明去查询 Book 表，后面的参数全部为 `null`，这就表示希望查询这张表中的所有数据。那么使用 LitePal 如何完成同样的功能呢？非常简单，只需要这样写：

```
List<Book> books = DataSupport.findAll(Book.class);
```

怎么样，代码是不是简单易懂多了？没有冗长的参数列表，只需要调用一下 `findAll()`方法，然后通过 `Book.class` 参数指定查询 Book 表就可以。另外，`findAll()`方法的返回值是一个 Book 类型的 List 集合，也就是说，我们不用像之前那样再通过 `Cursor` 对象一行行去取值了，LitePal 已经自动帮我们完成了赋值操作。

下面通过一个完整的例子来实践一下吧，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        Button queryButton = (Button) findViewById(R.id.query_data);
        queryButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                List<Book> books = DataSupport.findAll(Book.class);
                for (Book book: books) {
                    Log.d("MainActivity", "book name is " + book.getName());
                    Log.d("MainActivity", "book author is " + book.getAuthor());
                    Log.d("MainActivity", "book pages is " + book.getPages());
                    Log.d("MainActivity", "book price is " + book.getPrice());
                    Log.d("MainActivity", "book press is " + book.getPress());
                }
            }
        });
    }

}
```

查询的那段代码刚刚已经解释过了，接下来就是遍历 List 集合中的 Book 对象，并将其中的信息全部打印出来。现在重新运行一下程序，点击 Query data 按钮，然后查看 logcat 的打印内容，结果如图 6.37 所示。

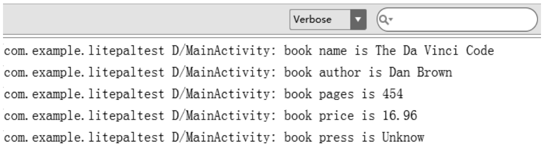


图 6.37　打印查询到的数据

Book 表中只剩下一条数据，由此可见，我们已经将这条数据成功查询出来了。

除了 `findAll()`方法之外，LitePal 还提供了很多其他非常有用的查询 API。比如我们想要查询 Book 表中的第一条数据就可以这样写：

```
Book firstBook = DataSupport.findFirst(Book.class);
```

查询 Book 表中的最后一条数据就可以这样写：

```
Book lastBook = DataSupport.findLast(Book.class);
```

我们还可以通过连缀查询来定制更多的查询功能。

- `select()`方法用于指定查询哪几列的数据，对应了 SQL 当中的 `select` 关键字。比如只查 `name` 和 `author` 这两列的数据，就可以这样写：

```
List<Book> books = DataSupport.select("name", "author").find(Book.class);
```

- `where()`方法用于指定查询的约束条件，对应了 SQL 当中的 `where` 关键字。比如只查页数大于 400 的数据，就可以这样写：

```
List<Book> books = DataSupport.where("pages > ?", "400").find(Book.class);
```

- `order()`方法用于指定结果的排序方式，对应了 SQL 当中的 `order by` 关键字。比如将查询结果按照书价从高到低排序，就可以这样写：

```
List<Book> books = DataSupport.order("price desc").find(Book.class);
```

  其中 `desc` 表示降序排列，`asc` 或者不写表示升序排列。

- `limit()`方法用于指定查询结果的数量，比如只查表中的前 3 条数据，就可以这样写：

```
List<Book> books = DataSupport.limit(3).find(Book.class);
```

- `offset()`方法用于指定查询结果的偏移量，比如查询表中的第 2 条、第 3 条、第 4 条数据，就可以这样写：

```
List<Book> books = DataSupport.limit(3).offset(1).find(Book.class);
```

由于 `limit(3)`查询到的是前 3 条数据，这里我们再加上 `offset(1)`进行一个位置的偏移，就能实现查询第 2 条、第 3 条、第 4 条数据的功能了。`limit()`和 `offset()`方法共同对应了 SQL 当中的 `limit` 关键字。

当然，你还可以对这 5 个方法进行任意的连缀组合，来完成一个比较复杂的查询操作：

```
List<Book> books = DataSupport.select("name", "author", "pages")
                              .where("pages > ?", "400")
                              .order("pages")
                              .limit(10)
                              .offset(10)
                              .find(Book.class);
```

这段代码就表示，查询 Book 表中第 11~20 条满足页数大于 400 这个条件的 `name`、`author` 和 `pages` 这 3 列数据，并将查询结果按照页数升序排列。

怎么样？是不是感觉 LitePal 的查询功能非常强大，并且代码明显更加简洁？我们需要用到一个方法的时候直接连缀一下就可以了，不需要的话就可以不写，而不是像之前的 `query()`方法，不管需不需要用到，都必须要传固定的参数进去才行。

关于 LitePal 的查询 API 差不多就介绍到这里，这些 API 已经足够我们应对绝大多数场景的查询需求了。当前，如果你实在有一些特殊需求，上述的 API 都满足不了你的时候，LitePal 仍然支持使用原生的 SQL 来进行查询：

```
Cursor c = DataSupport.findBySQL("select * from Book where pages > ? and price < ?",
"400",  "20");
```

调用 `DataSupport.findBySQL()`方法来进行原生查询，其中第一个参数用于指定 SQL 语句，后面的参数用于指定占位符的值。注意 `findBySQL()`方法返回的是一个 `Cursor` 对象，接下来你还需要通过之前所学的老方式将数据一一取出才行。

6.6　小结与点评

经过了这一章漫长的学习，我们终于可以缓解一下疲劳，对本章所学的知识进行梳理和总结了。本章主要是对 Android 常用的数据持久化方式进行了详细的讲解，包括文件存储、SharedPreferences 存储以及数据库存储。其中文件适用于存储一些简单的文本数据或者二进制数据，SharedPreferences 适用于存储一些键值对，而数据库则适用于存储那些复杂的关系型数据。虽然目前你已经掌握了这 3 种数据持久化方式的用法，但是能够根据项目的实际需求来选择最合适的方式也是你未来需要继续探索的。

那么正如上一章小结里提到的，既然现在我们已经掌握了 Android 中的数据持久化技术，接下来就应该继续学习 Android 中剩余的四大组件了。放松一下自己，然后一起踏上内容提供器的学习之旅吧。

第 7 章

跨程序共享数据——探究内容提供器

在上一章中我们学了 Android 数据持久化的技术，包括文件存储、SharedPreferences 存储以及数据库存储。不知道你有没有发现，使用这些持久化技术所保存的数据都只能在当前应用程序中访问。虽然文件和 SharedPreferences 存储中提供了 MODE_WORLD_READABLE 和 MODE_WORLD_WRITEABLE 这两种操作模式，用于供给其他的应用程序访问当前应用的数据，但这两种模式在 Android 4.2 版本中都已被废弃了。为什么呢？因为 Android 官方已经不再推荐使用这种方式来实现跨程序数据共享的功能，而是应该使用更加安全可靠的内容提供器技术。

可能你会有些疑惑，为什么要将我们程序中的数据共享给其他程序呢？当然，这个是要视情况而定的，比如说账号和密码这样的隐私数据显然是不能共享给其他程序的，不过一些可以让其他程序进行二次开发的基础性数据，我们还是可以选择将其共享的。例如系统的电话簿程序，它的数据库中保存了很多的联系人信息，如果这些数据都不允许第三方的程序进行访问的话，恐怕很多应用的功能都要大打折扣了。除了电话簿之外，还有短信、媒体库等程序都实现了跨程序数据共享的功能，而使用的技术当然就是内容提供器了，下面我们就来对这一技术进行深入的探讨。

7.1 内容提供器简介

内容提供器（Content Provider）主要用于在不同的应用程序之间实现数据共享的功能，它提供了一套完整的机制，允许一个程序访问另一个程序中的数据，同时还能保证被访数据的安全性。目前，使用内容提供器是 Android 实现跨程序共享数据的标准方式。

不同于文件存储和 SharedPreferences 存储中的两种全局可读写操作模式，内容提供器可以选择只对哪一部分数据进行共享，从而保证我们程序中的隐私数据不会有泄漏的风险。

不过在正式开始学习内容提供器之前，我们需要先掌握另外一个非常重要的知识——Android 运行时权限，因为待会的内容提供器示例中会使用到运行时权限的功能。当然不光是

内容提供器，以后我们的开发过程中也会经常使用到运行时权限，因此你必须能够牢牢掌握它才行。

7.2 运行时权限

Android 的权限机制并不是什么新鲜事物，从系统的第一个版本开始就已经存在了。但其实之前 Android 的权限机制在保护用户安全和隐私等方面起到的作用比较有限，尤其是一些大家都离不开的常用软件，非常容易“店大欺客”。为此，Android 开发团队在 Android 6.0 系统中引用了运行时权限这个功能，从而更好地保护了用户的安全和隐私，那么本节我们就来详细学习一下这个 6.0 系统中引入的新特性。

7.2.1 Android 权限机制详解

首先来回顾一下过去 Android 的权限机制是什么样的。我们在第 5 章写 BroadcastTest 项目的时候第一次接触了 Android 权限相关的内容，当时为了要访问系统的网络状态以及监听开机广播，于是在 AndroidManifest.xml 文件中添加了这样两句权限声明：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest">

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    ...
</manifest>
```

因为访问系统的网络状态以及监听开机广播涉及了用户设备的安全性，因此必须在 AndroidManifest.xml 中加入权限声明，否则我们的程序就会崩溃。

那么现在问题来了，加入了这两句权限声明后，对于用户来说到底有什么影响呢？为什么这样就可以保护用户设备的安全性了呢？

其实用户主要在以下两个方面得到了保护，一方面，如果用户在低于 6.0 系统的设备上安装该程序，会在安装界面给出如图 7.1 所示的提醒。这样用户就可以清楚地知晓该程序一共申请了哪些权限，从而决定是否要安装这个程序。



图 7.1　安装界面的权限提醒

另一方面，用户可以随时在应用程序管理界面查看任意一个程序的权限申请情况，如图 7.2 所示。这样该程序申请的所有权限就尽收眼底，什么都瞒不过用户的眼睛，以此保证应用程序不会出现各种滥用权限的情况。



图 7.2　管理界面的权限展示

这种权限机制的设计思路其实非常简单，就是用户如果认可你所申请的权限，那么就会安装你的程序，如果不认可你所申请的权限，那么拒绝安装就可以了。

但是理想是美好的，现实却很残酷，因为很多我们所离不开的常用软件普遍存在着滥用权限

的情况，不管到底用不用得到，反正先把权限申请了再说。比如说微信所申请的权限列表如图 7.3 所示。



图 7.3 微信的权限列表

这只是微信所申请的一半左右的权限，因为权限太多一屏截不下来。其中有一些权限我并不认可，比如微信为什么要读取我手机的短信和彩信？但是我不认可又能怎样，难道我拒绝安装微信？没错，这种例子比比皆是，当一些软件已经让我们产生依赖的时候就会容易“店大欺客”，反正这个权限我就是要了，你自己看着办吧！

Android 开发团队当然也意识到了这个问题，于是在 6.0 系统中加入了运行时权限功能。也就是说，用户不需要在安装软件的时候一次性授权所有申请的权限，而是可以在软件的使用过程中再对某一项权限申请进行授权。比如说一款相机应用在运行时申请了地理位置定位权限，就算我拒绝了这个权限，但是我应该仍然可以使用这个应用的其他功能，而不是像之前那样直接无法安装它。

当然，并不是所有权限都需要在运行时申请，对于用户来说，不停地授权也很烦琐。Android 现在将所有的权限归成了两类，一类是普通权限，一类是危险权限。准确地讲，其实还有第三类特殊权限，不过这种权限使用得很少，因此不在本书的讨论范围之内。普通权限指的是那些不会直接威胁到用户的安全和隐私的权限，对于这部分权限申请，系统会自动帮我们进行授权，而不需要用户再去手动操作了，比如在 BroadcastTest 项目中申请的两个权限就是普通权限。危险权限则表示那些可能会触及用户隐私或者对设备安全性造成影响的权限，如获取设备联系人信息、定位设备的地理位置等，对于这部分权限申请，必须要由用户手动点击授权才可以，否则程序就无法使用相应的功能。

但是 Android 中有一共有上百种权限，我们怎么从中区分哪些是普通权限，哪些是危险权限

呢？其实并没有那么难，因为危险权限总共就那么几个，除了危险权限之外，剩余的就都是普通权限了。下表列出了 Android 中所有的危险权限，一共是 9 组 24 个权限。

权限组名	权限名
CALENDAR	READ_CALENDAR WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE CALL_PHONE READ_CALL_LOG WRITE_CALL_LOG ADD_VOICEMAIL USE_SIP PROCESS_OUTGOING_CALLS
SENSORS	BODY_SENSORS
SMS	SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

这张表格你看起来可能并不会那么轻松，因为里面的权限全都是你没使用过的。不过没有关系，你并不需要了解表格中每个权限的作用，只要把它当成一个参照表来查看就行了。每当要使用一个权限时，可以先到这张表中来查一下，如果是属于这张表中的权限，那么就需要进行运行时权限处理，如果不在这张表中，那么只需要在 AndroidManifest.xml 文件中添加一下权限声明就可以了。

另外注意一下，表格中每个危险权限都属于一个权限组，我们在进行运行时权限处理时使用的是权限名，但是用户一旦同意授权了，那么该权限所对应的权限组中所有的其他权限也会同时被授权。

访问 http://developer.android.google.cn/reference/android/Manifest.permission.html 可以查看 Android 系统中完整的权限列表。

好了，关于 Android 权限机制的内容就讲这么多，理论知识你已经了解得非常充足了。接下来我们就学习一下到底如何在程序运行的时候申请权限。

7.2.2 在程序运行时申请权限

首先新建一个 RuntimePermissionTest 项目，我们就在这个项目的基础上来学习运行时权限的使用方法。在开始动手之前还需要考虑一下到底要申请什么权限，其实刚才表中列出的所有权限都是可以申请的，这里简单起见我们就使用 `CALL_PHONE` 这个权限来作为本小节中的示例吧。

`CALL_PHONE` 这个权限是编写拨打电话功能的时候需要声明的，因为拨打电话会涉及用户手机的资费问题，因而被列为了危险权限。在 Android 6.0 系统出现之前，拨打电话功能的实现其实非常简单，修改 activity_main.xml 布局文件，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/make_call"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Make Call" />

</LinearLayout>
```

我们在布局文件中只是定义了一个按钮，当点击按钮时就去触发拨打电话的逻辑。接着修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button makeCall = (Button) findViewById(R.id.make_call);
        makeCall.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try {
                    Intent intent = new Intent(Intent.ACTION_CALL);
                    intent.setData(Uri.parse("tel:10086"));
                    startActivity(intent);
                } catch (SecurityException e) {
                    e.printStackTrace();
                }
            }
        });
    }

}
```

可以看到，在按钮的点击事件中，我们构建了一个隐式 Intent ，Intent 的 action 指定为 `Intent.ACTION_ CALL`，这是一个系统内置的打电话的动作，然后在 data 部分指定了协议是 tel，号码是 10086。其实这部分代码我们在 2.3.3 小节中就已经见过了，只不过当时指定的 action 是 `Intent.ACTION_DIAL`，表示打开拨号界面，这个是不需要声明权限的，而 `Intent.ACTION_ CALL` 则可以直接拨打电话，因此必须声明权限。另外为了防止程序崩溃，我们将所有操作都放在了异常捕获代码块当中。

那么接下来修改 AndroidManifest.xml 文件，在其中声明如下权限：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.runtimepermissiontest">

    <uses-permission android:name="android.permission.CALL_PHONE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
    </application>

</manifest>
```

这样我们就将拨打电话的功能成功实现了，并且在低于 Android 6.0 系统的手机上都是可以正常运行的，但是如果我们在 6.0 或者更高版本系统的手机上运行，点击 Make Call 按钮就没有任何效果，这时观察 logcat 中的打印日志，你会看到如图 7.4 所示的错误信息。

```
java.lang.SecurityException: Permission Denial: starting Intent { act=android.intent.action.CALL
    at android.os.Parcel.readException(Parcel.java:1599)
    at android.os.Parcel.readException(Parcel.java:1552)
    at android.app.ActivityManagerProxy.startActivity(ActivityManagerNative.java:2658)
    at android.app.Instrumentation.execStartActivity(Instrumentation.java:1507)
    at android.app.Activity.startActivityForResult(Activity.java:3917)
    at android.app.Activity.startActivityForResult(Activity.java:3877)
    at android.support.v4.app.FragmentActivity.startActivityForResult(FragmentActivity.java:843)
    at android.app.Activity.startActivity(Activity.java:4200)
    at android.app.Activity.startActivity(Activity.java:4168)
    at com.example.runtimepermissiontest.MainActivity$1.onClick(MainActivity.java:29)
```

图 7.4　错误日志信息

错误信息中提醒我们“Permission Denial”，可以看出，是由于权限被禁止所导致的，因为 6.0 及以上系统在使用危险权限时都必须进行运行时权限处理。

那么下面我们就来尝试修复这个问题，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button makeCall = (Button) findViewById(R.id.make_call);
        makeCall.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.
                    permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
                    ActivityCompat.requestPermissions(MainActivity.this, new
                        String[]{ Manifest.permission.CALL_PHONE }, 1);
                } else {
                    call();
                }
            }
        });
    }

    private void call() {
        try {
            Intent intent = new Intent(Intent.ACTION_CALL);
            intent.setData(Uri.parse("tel:10086"));
            startActivity(intent);
        } catch (SecurityException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
        int[] grantResults) {
        switch (requestCode) {
            case 1:
                if (grantResults.length > 0 && grantResults[0] == PackageManager.
                    PERMISSION_GRANTED) {
                    call();
                } else {
                    Toast.makeText(this, "You denied the permission", Toast.LENGTH_
                        SHORT).show();
                }
                break;
            default:
        }
    }
}
```

上面的代码将运行时权限的完整流程都覆盖了，下面我们来具体解析一下。说白了，运行时权限的核心就是在程序运行过程中由用户授权我们去执行某些危险操作，程序是不可以擅自做主去执行这些危险操作的。因此，第一步就是要先判断用户是不是已经给过我们授权了，借助的是 ContextCompat.checkSelfPermission()方法。checkSelfPermission()方法接收两个参数，第一个参数是 Context，这个没什么好说的，第二个参数是具体的权限名，比如打电话的权限名

就是 Manifest.permission.CALL_PHONE，然后我们使用方法的返回值和 PackageManager.PERMISSION_GRANTED 做比较，相等就说明用户已经授权，不等就表示用户没有授权。

如果已经授权的话就简单了，直接去执行拨打电话的逻辑操作就可以了，这里我们把拨打电话的逻辑封装到了 call()方法当中。如果没有授权的话，则需要调用 ActivityCompat.requestPermissions()方法来向用户申请授权，requestPermissions()方法接收 3 个参数，第一个参数要求是 Activity 的实例，第二个参数是一个 String 数组，我们把要申请的权限名放在数组中即可，第三个参数是请求码，只要是唯一值就可以了，这里传入了 1。

调用完了 requestPermissions()方法之后，系统会弹出一个权限申请的对话框，然后用户可以选择同意或拒绝我们的权限申请，不论是哪种结果，最终都会回调到 onRequestPermissionsResult()方法中，而授权的结果则会封装在 grantResults 参数当中。这里我们只需要判断一下最后的授权结果，如果用户同意的话就调用 call()方法来拨打电话，如果用户拒绝的话我们只能放弃操作，并且弹出一条失败提示。

现在重新运行一下程序，并点击 Make Call 按钮，效果如图 7.5 所示。

由于用户还没有授权过我们拨打电话权限，因此第一次运行会弹出这样一个权限申请的对话框，用户可以选择同意或者拒绝，比如说这里点击了 DENY，结果如图 7.6 所示。

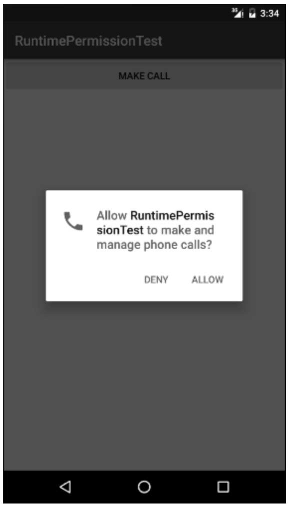


图 7.5　申请电话权限对话框



图 7.6　用户拒绝了权限申请

由于用户没有同意授权，我们只能弹出一个操作失败的提示。下面我们再次点击 Make Call 按钮，仍然会弹出权限申请的对话框，这次点击 ALLOW，结果如图 7.7 所示。

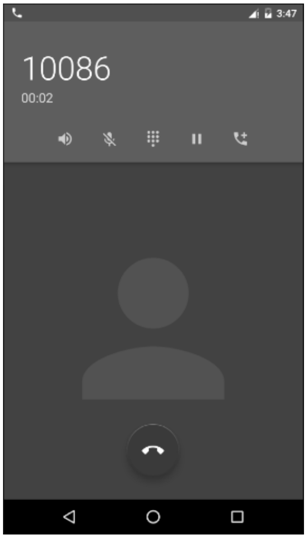


图 7.7　拨打电话界面

可以看到，这次我们就成功进入到拨打电话界面了，并且由于用户已经完成了授权操作，之后再点击 Make Call 按钮就不会再弹出权限申请对话框了，而是可以直接拨打电话。那可能你会担心，万一以后我又后悔了怎么办？没有关系，用户随时都可以将授予程序的危险权限进行关闭，进入 Settings → Apps → RuntimePermissionTest → Permissions，界面如图 7.8 所示。

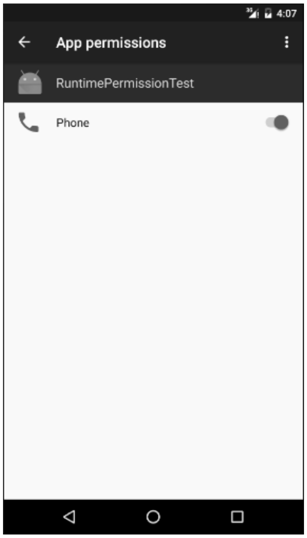


图 7.8　应用程序权限管理界面

在这里我们就可以对任何授予过的危险权限进行关闭了。

好了，关于运行时权限的内容就讲到这里，现在你已经有能力处理 Android 上各种关于权限的问题了，下面我们就来进入本章的正题——内容提供器。

7.3　访问其他程序中的数据

内容提供器的用法一般有两种，一种是使用现有的内容提供器来读取和操作相应程序中的数据，另一种是创建自己的内容提供器给我们程序的数据提供外部访问接口。那么接下来我们就一个一个开始学习吧，首先从使用现有的内容提供器开始。

如果一个应用程序通过内容提供器对其数据提供了外部访问接口，那么任何其他的应用程序就都可以对这部分数据进行访问。Android 系统中自带的电话簿、短信、媒体库等程序都提供了类似的访问接口，这就使得第三方应用程序可以充分地利用这部分数据来实现更好的功能。下面我们就来看一看，内容提供器到底是如何使用的。

7.3.1　ContentResolver 的基本用法

对于每一个应用程序来说，如果想要访问内容提供器中共享的数据，就一定要借助 ContentResolver 类，可以通过 Context 中的 `getContentResolver()`方法获取到该类的实例。ContentResolver 中提供了一系列的方法用于对数据进行 CRUD 操作，其中 `insert()`方法用于添加数据，`update()`方法用于更新数据，`delete()`方法用于删除数据，`query()`方法用于查询数据。有没有似曾相识的感觉？没错，SQLiteDatabase 中也是使用这几个方法来进行 CRUD 操作的，只不过它们在方法参数上稍微有一些区别。

不同于 SQLiteDatabase，ContentResolver 中的增删改查方法都是不接收表名参数的，而是使用一个 `Uri` 参数代替，这个参数被称为内容 URI。内容 URI 给内容提供器中的数据建立了唯一标识符，它主要由两部分组成：authority 和 path。authority 是用于对不同的应用程序做区分的，一般为了避免冲突，都会采用程序包名的方式来进行命名。比如某个程序的包名是 com.example.app，那么该程序对应的 authority 就可以命名为 com.example.app. provider。path 则是用于对同一应用程序中不同的表做区分的，通常都会添加到 authority 的后面。比如某个程序的数据库里存在两张表：table1 和 table2，这时就可以将 path 分别命名为/table1 和/table2，然后把 authority 和 path 进行组合，内容 URI 就变成了 com.example.app.provider/table1 和 com.example.app.provider/table2。不过，目前还很难辨认出这两个字符串就是两个内容 URI，我们还需要在字符串的头部加上协议声明。因此，内容 URI 最标准的格式写法如下：

```
content://com.example.app.provider/table1
content://com.example.app.provider/table2
```

有没有发现，内容 URI 可以非常清楚地表达出我们想要访问哪个程序中哪张表里的数据。也正是因此，ContentResolver 中的增删改查方法才都接收 `Uri` 对象作为参数，因为如果使用表名的话，系统将无法得知我们期望访问的是哪个应用程序里的表。

在得到了内容 URI 字符串之后，我们还需要将它解析成 Uri 对象才可以作为参数传入。解析的方法也相当简单，代码如下所示：

```
Uri uri = Uri.parse("content://com.example.app.provider/table1")
```

只需要调用 Uri.parse()方法，就可以将内容 URI 字符串解析成 Uri 对象了。

现在我们就可以使用这个 Uri 对象来查询 table1 表中的数据了，代码如下所示：

```
Cursor cursor = getContentResolver().query(
    uri,
    projection,
    selection,
    selectionArgs,
    sortOrder);
```

这些参数和 SQLiteDatabase 中 query()方法里的参数很像，但总体来说要简单一些，毕竟这是在访问其他程序中的数据，没必要构建过于复杂的查询语句。下表对使用到的这部分参数进行了详细的解释。

query()方法参数	对应SQL部分	描　　述
uri	from table_name	指定查询某个应用程序下的某一张表
projection	select column1, column2	指定查询的列名
selection	where column = value	指定where的约束条件
selectionArgs	-	为where中的占位符提供具体的值
sortOrder	order by column1, column2	指定查询结果的排序方式

查询完成后返回的仍然是一个 Cursor 对象，这时我们就可以将数据从 Cursor 对象中逐个读取出来了。读取的思路仍然是通过移动游标的位置来遍历 Cursor 的所有行，然后再取出每一行中相应列的数据，代码如下所示：

```
if (cursor != null) {
    while (cursor.moveToNext()) {
        String column1 = cursor.getString(cursor.getColumnIndex("column1"));
        int column2 = cursor.getInt(cursor.getColumnIndex("column2"));
    }
    cursor.close();
}
```

掌握了最难的查询操作，剩下的增加、修改、删除操作就更不在话下了。我们先来看看如何向 table1 表中添加一条数据，代码如下所示：

```
ContentValues values = new ContentValues();
values.put("column1", "text");
values.put("column2", 1);
getContentResolver().insert(uri, values);
```

可以看到，仍然是将待添加的数据组装到 ContentValues 中，然后调用 ContentResolver 的

insert()方法，将 Uri 和 ContentValues 作为参数传入即可。

现在如果我们想要更新这条新添加的数据，把 column1 的值清空，可以借助 ContentResolver 的 update()方法实现，代码如下所示：

```
ContentValues values = new ContentValues();
values.put("column1", "");
getContentResolver().update(uri, values, "column1 = ? and column2 = ?", new
String[] {"text", "1"});
```

注意上述代码使用了 selection 和 selectionArgs 参数来对想要更新的数据进行约束，以防止所有的行都会受影响。

最后，可以调用 ContentResolver 的 delete()方法将这条数据删除掉，代码如下所示：

```
getContentResolver().delete(uri, "column2 = ?", new String[] { "1" });
```

到这里为止，我们就把 ContentResolver 中的增删改查方法全部学完了。是不是感觉一看就懂？因为这些知识早在上一章中学习 SQLiteDatabase 的时候你就已经掌握了，所需特别注意的就只有 uri 这个参数而已。那么接下来，我们就利用目前所学的知识，看一看如何读取系统电话簿中的联系人信息。

7.3.2　读取系统联系人

由于我们之前一直使用的都是模拟器，电话簿里面并没有联系人存在，所以现在需要自己手动添加几个，以便稍后进行读取。打开电话簿程序，界面如图 7.9 所示。

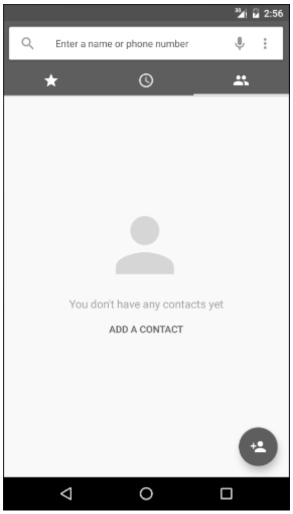


图 7.9　电话薄程序主界面

可以看到，目前电话簿里是没有任何联系人的，我们可以通过点击 ADD A CONTACT 按钮来对联系人进行创建。这里就先创建两个联系人吧，分别填入他们的姓名和手机号，如图 7.10 所示。

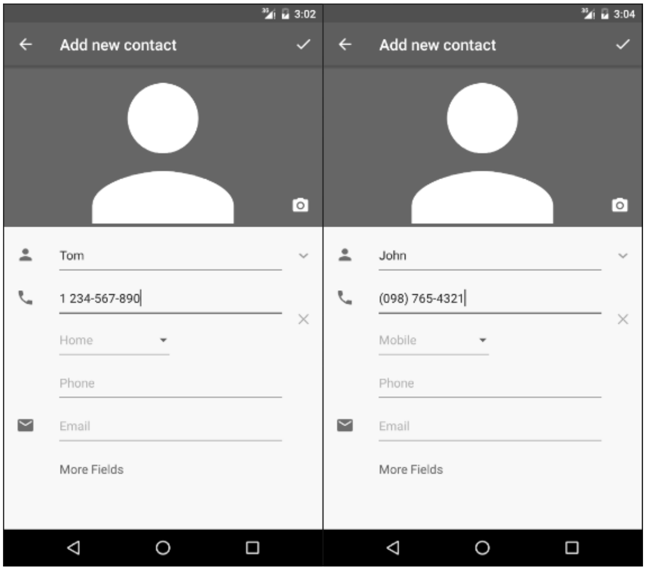


图 7.10　添加两个联系人

这样准备工作就做好了，现在新建一个 ContactsTest 项目，让我们开始动手吧。

首先还是来编写一下布局文件，这里我们希望读取出来的联系人信息能够在 ListView 中显示，因此，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <ListView
        android:id="@+id/contacts_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
    </ListView>

</LinearLayout>
```

简单起见，LinearLayout 里就只放置了一个 ListView。这里使用 ListView 而不是 RecyclerView，是因为我们要将关注的重点放在读取系统联系人上面，如果使用 RecyclerView 的话，代码偏多，会容易让我们找不着重点。

接着修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ArrayAdapter<String> adapter;

    List<String> contactsList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView contactsView = (ListView) findViewById(R.id.contacts_view);
        adapter = new ArrayAdapter<String>(this, android.R.layout. simple_list_
            item_1, contactsList);
        contactsView.setAdapter(adapter);
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_
            CONTACTS) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, new String[]{ Manifest.
                permission.READ_CONTACTS }, 1);
        } else {
            readContacts();
        }
    }

    private void readContacts() {
        Cursor cursor = null;
        try {
            // 查询联系人数据
            cursor = getContentResolver().query(ContactsContract.CommonDataKinds.
                Phone.CONTENT_URI, null, null, null, null);
            if (cursor != null) {
                while (cursor.moveToNext()) {
                    // 获取联系人姓名
                    String displayName = cursor.getString(cursor.getColumnIndex
                       (ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME));
                    // 获取联系人手机号
                    String number = cursor.getString(cursor.getColumnIndex
                       (ContactsContract.CommonDataKinds.Phone.NUMBER));
                    contactsList.add(displayName + "\n" + number);
                }
                adapter.notifyDataSetChanged();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (cursor != null) {
                cursor.close();
            }
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
        int[] grantResults) {
        switch (requestCode) {
```

```
            case 1:
                if (grantResults.length > 0 && grantResults[0] == PackageManager.
                    PERMISSION_GRANTED) {
                    readContacts();
                } else {
                    Toast.makeText(this, "You denied the permission", Toast.LENGTH_
                        SHORT).show();
                }
                break;
            default:
        }
    }

}
```

在 onCreate()方法中，我们首先获取了 ListView 控件的实例，并给它设置好了适配器，然后开始调用运行时权限的处理逻辑，因为 READ_CONTACTS 权限是属于危险权限的。关于运行时权限的处理流程相信你已经熟练掌握了，这里我们在用户授权之后调用 readContacts()方法来读取系统联系人信息。

下面重点看一下 readContacts()方法，可以看到，这里使用了 ContentResolver 的 query()方法来查询系统的联系人数据。不过传入的 Uri 参数怎么有些奇怪啊？为什么没有调用 Uri.parse()方法去解析一个内容 URI 字符串呢？这是因为 ContactsContract.CommonDataKinds.Phone 类已经帮我们做好了封装，提供了一个 CONTENT_URI 常量，而这个常量就是使用 Uri.parse()方法解析出来的结果。接着我们对 Cursor 对象进行遍历，将联系人姓名和手机号这些数据逐个取出，联系人姓名这一列对应的常量是 ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME，联系人手机号这一列对应的常量是 ContactsContract.CommonDataKinds.Phone.NUMBER。两个数据都取出之后，将它们进行拼接，并且在中间加上换行符，然后将拼接后的数据添加到 ListView 的数据源里，并通知刷新一下 ListView。最后千万不要忘记将 Cursor 对象关闭掉。

这样就结束了吗？还差一点点，读取系统联系人的权限千万不能忘记声明。修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.contactstest">

    <uses-permission android:name="android.permission.READ_CONTACTS" />
    ...
</manifest>
```

加入了 android.permission.READ_CONTACTS 权限，这样我们的程序就可以访问到系统的联系人数据了。现在才算是大功告成了，让我们来运行一下程序吧，效果如图 7.11 所示。

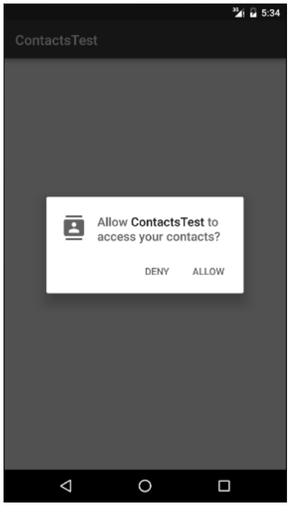


图 7.11　申请访问联系人权限对话框

首先弹出了申请访问联系人权限的对话框，我们点击 ALLOW，然后结果如图 7.12 所示。



图 7.12　展示系统联系人信息

刚刚添加的两个联系人的数据都成功读取出来了！这说明跨程序访问数据的功能确实是实现了。

7.4　创建自己的内容提供器

在上一节当中，我们学习了如何在自己的程序中访问其他应用程序的数据。总体来说思路还

是非常简单的，只需要获取到该应用程序的内容 URI，然后借助 ContentResolver 进行 CRUD 操作就可以了。可是你有没有想过，那些提供外部访问接口的应用程序都是如何实现这种功能的呢？它们又是怎样保证数据的安全性，使得隐私数据不会泄漏出去？学习完本节的知识后，你的疑惑将会被一一解开。

7.4.1 创建内容提供器的步骤

前面已经提到过，如果想要实现跨程序共享数据的功能，官方推荐的方式就是使用内容提供器，可以通过新建一个类去继承 ContentProvider 的方式来创建一个自己的内容提供器。ContentProvider 类中有 6 个抽象方法，我们在使用子类继承它的时候，需要将这 6 个方法全部重写。新建 MyProvider 继承自 ContentProvider，代码如下所示：

```
public class MyProvider extends ContentProvider {

    @Override
    public boolean onCreate() {
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[]
        selectionArgs, String sortOrder) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection, String[]
        selectionArgs) {
        return 0;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }

}
```

在这 6 个方法中，相信大多数你都已经非常熟悉了，我再来简单介绍一下吧。

1. onCreate()

初始化内容提供器的时候调用。通常会在这里完成对数据库的创建和升级等操作，返回 true 表示内容提供器初始化成功，返回 false 则表示失败。

2. query()

从内容提供器中查询数据。使用 uri 参数来确定查询哪张表，projection 参数用于确定查询哪些列，selection 和 selectionArgs 参数用于约束查询哪些行，sortOrder 参数用于对结果进行排序，查询的结果存放在 Cursor 对象中返回。

3. insert()

向内容提供器中添加一条数据。使用 uri 参数来确定要添加到的表，待添加的数据保存在 values 参数中。添加完成后，返回一个用于表示这条新记录的 URI。

4. update()

更新内容提供器中已有的数据。使用 uri 参数来确定更新哪一张表中的数据，新数据保存在 values 参数中，selection 和 selectionArgs 参数用于约束更新哪些行，受影响的行数将作为返回值返回。

5. delete()

从内容提供器中删除数据。使用 uri 参数来确定删除哪一张表中的数据，selection 和 selectionArgs 参数用于约束删除哪些行，被删除的行数将作为返回值返回。

6. getType()

根据传入的内容 URI 来返回相应的 MIME 类型。

可以看到，几乎每一个方法都会带有 Uri 这个参数，这个参数也正是调用 ContentResolver 的增删改查方法时传递过来的。而现在，我们需要对传入的 Uri 参数进行解析，从中分析出调用方期望访问的表和数据。

回顾一下，一个标准的内容 URI 写法是这样的：

```
content://com.example.app.provider/table1
```

这就表示调用方期望访问的是 com.example.app 这个应用的 table1 表中的数据。除此之外，我们还可以在这个内容 URI 的后面加上一个 id，如下所示：

```
content://com.example.app.provider/table1/1
```

这就表示调用方期望访问的是 com.example.app 这个应用的 table1 表中 id 为 1 的数据。

内容 URI 的格式主要就只有以上两种，以路径结尾就表示期望访问该表中所有的数据，以 id 结尾就表示期望访问该表中拥有相应 id 的数据。我们可以使用通配符的方式来分别匹配这两种格式的内容 URI，规则如下。

- *：表示匹配任意长度的任意字符。
- #：表示匹配任意长度的数字。

所以，一个能够匹配任意表的内容 URI 格式就可以写成：

```
content://com.example.app.provider/*
```

而一个能够匹配 table1 表中任意一行数据的内容 URI 格式就可以写成：

```
content://com.example.app.provider/table1/#
```

接着，我们再借助 UriMatcher 这个类就可以轻松地实现匹配内容 URI 的功能。UriMatcher 中提供了一个 addURI()方法，这个方法接收 3 个参数，可以分别把 authority、path 和一个自定义代码传进去。这样，当调用 UriMatcher 的 match()方法时，就可以将一个 Uri 对象传入，返回值是某个能够匹配这个 Uri 对象所对应的自定义代码，利用这个代码，我们就可以判断出调用方期望访问的是哪张表中的数据了。修改 MyProvider 中的代码，如下所示：

```
public class MyProvider extends ContentProvider {

    public static final int TABLE1_DIR = 0;

    public static final int TABLE1_ITEM = 1;

    public static final int TABLE2_DIR = 2;

    public static final int TABLE2_ITEM = 3;

    private static UriMatcher uriMatcher;

    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI("com.example.app.provider", "table1", TABLE1_DIR);
        uriMatcher.addURI("com.example.app.provider ", "table1/#", TABLE1_ITEM);
        uriMatcher.addURI("com.example.app.provider ", "table2", TABLE2_DIR);
        uriMatcher.addURI("com.example.app.provider ", "table2/#", TABLE2_ITEM);
    }

    ...

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[]
        selectionArgs, String sortOrder) {
        switch (uriMatcher.match(uri)) {
        case TABLE1_DIR:
            // 查询 table1 表中的所有数据
            break;
        case TABLE1_ITEM:
            // 查询 table1 表中的单条数据
            break;
```

```
            case TABLE2_DIR:
                // 查询table2表中的所有数据
                break;
            case TABLE2_ITEM:
                // 查询table2表中的单条数据
                break;
            default:
                break;
            }

            ...

        }

        ...
    }
```

可以看到，MyProvider 中新增了 4 个整型常量，其中 TABLE1_DIR 表示访问 table1 表中的所有数据，TABLE1_ITEM 表示访问 table1 表中的单条数据，TABLE2_DIR 表示访问 table2 表中的所有数据，TABLE2_ITEM 表示访问 table2 表中的单条数据。接着在静态代码块里我们创建了 UriMatcher 的实例，并调用 addURI()方法，将期望匹配的内容 URI 格式传递进去，注意这里传入的路径参数是可以使用通配符的。然后当 query()方法被调用的时候，就会通过 UriMatcher 的 match()方法对传入的 Uri 对象进行匹配，如果发现 UriMatcher 中某个内容 URI 格式成功匹配了该 Uri 对象，则会返回相应的自定义代码，然后我们就可以判断出调用方期望访问的到底是什么数据了。

上述代码只是以 query()方法为例做了个示范，其实 insert()、update()、delete()这几个方法的实现也是差不多的，它们都会携带 Uri 这个参数，然后同样利用 UriMatcher 的 match()方法判断出调用方期望访问的是哪张表，再对该表中的数据进行相应的操作就可以了。

除此之外，还有一个方法你会比较陌生，即 getType()方法。它是所有的内容提供器都必须提供的一个方法，用于获取 Uri 对象所对应的 MIME 类型。一个内容 URI 所对应的 MIME 字符串主要由 3 部分组成，Android 对这 3 个部分做了如下格式规定。

- 必须以 vnd 开头。
- 如果内容 URI 以路径结尾，则后接 android.cursor.dir/，如果内容 URI 以 id 结尾，则后接 android.cursor.item/。
- 最后接上 vnd.<authority>.<path>。

所以，对于 content://com.example.app.provider/table1 这个内容 URI，它所对应的 MIME 类型就可以写成：

```
vnd.android.cursor.dir/vnd.com.example.app.provider.table1
```

对于 content://com.example.app.provider/table1/1 这个内容 URI，它所对应的 MIME 类型就可以写成：

```
vnd.android.cursor.item/vnd.com.example.app.provider.table1
```

现在我们可以继续完善 MyProvider 中的内容了，这次来实现 getType()方法中的逻辑，代码如下所示：

```
public class MyProvider extends ContentProvider {

    ...

    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)) {
        case TABLE1_DIR:
            return "vnd.android.cursor.dir/vnd.com.example.app.provider.table1";
        case TABLE1_ITEM:
            return "vnd.android.cursor.item/vnd.com.example.app.provider.table1";
        case TABLE2_DIR:
            return "vnd.android.cursor.dir/vnd.com.example.app.provider.table2";
        case TABLE2_ITEM:
            return "vnd.android.cursor.item/vnd.com.example.app.provider.table2";
        default:
            break;
        }
        return null;
    }

}
```

到这里，一个完整的内容提供器就创建完成了，现在任何一个应用程序都可以使用 ContentResolver 来访问我们程序中的数据。那么前面所提到的，如何才能保证隐私数据不会泄漏出去呢？其实多亏了内容提供器的良好机制，这个问题在不知不觉中已经被解决了。因为所有的 CRUD 操作都一定要匹配到相应的内容 URI 格式才能进行的，而我们当然不可能向 UriMatcher 中添加隐私数据的 URI，所以这部分数据根本无法被外部程序访问到，安全问题也就不存在了。

好了，创建内容提供器的步骤你也已经清楚了，下面就来实战一下，真正体验一回跨程序数据共享的功能。

7.4.2 实现跨程序数据共享

简单起见，我们还是在上一章中 DatabaseTest 项目的基础上继续开发，通过内容提供器来给它加入外部访问接口。打开 DatabaseTest 项目，首先将 MyDatabaseHelper 中使用 Toast 弹出创建数据库成功的提示去除掉，因为跨程序访问时我们不能直接使用 Toast。然后创建一个内容提供器，右击 com.example.databasetest 包→New→Other→Content Provider，会弹出如图 7.13 所示的窗口。

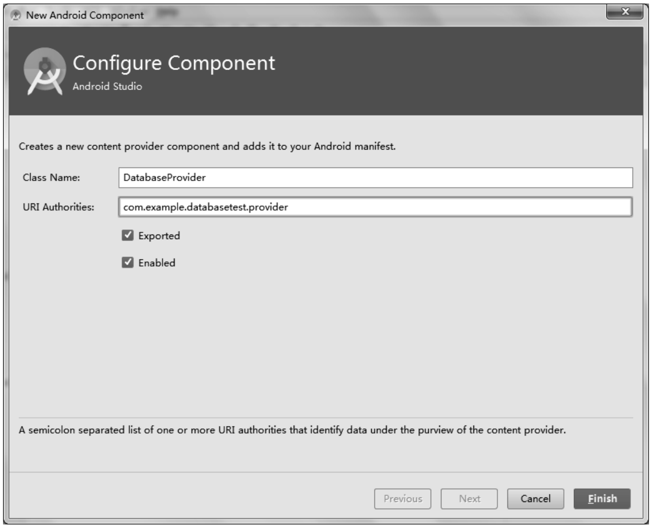


图 7.13　创建内容提供器的窗口

可以看到，这里我们将内容提供器命名为 DatabaseProvider，authority 指定为 com.example.databasetest.provider，Exported 属性表示是否允许外部程序访问我们的内容提供器，Enabled 属性表示是否启用这个内容提供器。将两个属性都勾中，点击 Finish 完成创建。

接着我们修改 DatabaseProvider 中的代码，如下所示：

```
public class DatabaseProvider extends ContentProvider {

    public static final int BOOK_DIR = 0;

    public static final int BOOK_ITEM = 1;

    public static final int CATEGORY_DIR = 2;

    public static final int CATEGORY_ITEM = 3;

    public static final String AUTHORITY = "com.example.databasetest.provider";

    private static UriMatcher uriMatcher;

    private MyDatabaseHelper dbHelper;

    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(AUTHORITY, "book", BOOK_DIR);
        uriMatcher.addURI(AUTHORITY, "book/#", BOOK_ITEM);
        uriMatcher.addURI(AUTHORITY, "category", CATEGORY_DIR);
        uriMatcher.addURI(AUTHORITY, "category/#", CATEGORY_ITEM);
    }
```

```
    @Override
    public boolean onCreate() {
        dbHelper = new MyDatabaseHelper(getContext(), "BookStore.db", null, 2);
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[]
        selectionArgs, String sortOrder) {
        // 查询数据
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        Cursor cursor = null;
        switch (uriMatcher.match(uri)) {
            case BOOK_DIR:
                cursor = db.query("Book", projection, selection, selectionArgs, null,
                    null, sortOrder);
                break;
            case BOOK_ITEM:
                String bookId = uri.getPathSegments().get(1);
                cursor = db.query("Book", projection, "id = ?", new String[] { bookId },
                    null, null, sortOrder);
                break;
            case CATEGORY_DIR:
                cursor = db.query("Category", projection, selection, selectionArgs,
                    null, null, sortOrder);
                break;
            case CATEGORY_ITEM:
                String categoryId = uri.getPathSegments().get(1);
                cursor = db.query("Category", projection, "id = ?", new String[]
                    { categoryId }, null, null, sortOrder);
                break;
            default:
                break;
        }
        return cursor;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // 添加数据
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        Uri uriReturn = null;
        switch (uriMatcher.match(uri)) {
            case BOOK_DIR:
            case BOOK_ITEM:
                long newBookId = db.insert("Book", null, values);
                uriReturn = Uri.parse("content://" + AUTHORITY + "/book/" +
                   newBookId);
                break;
            case CATEGORY_DIR:
            case CATEGORY_ITEM:
                long newCategoryId = db.insert("Category", null, values);
                uriReturn = Uri.parse("content://" + AUTHORITY + "/category/" +
                    newCategoryId);
```

```
                break;
            default:
                break;
        }
        return uriReturn;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection, String[]
        selectionArgs) {
        // 更新数据
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        int updatedRows = 0;
        switch (uriMatcher.match(uri)) {
            case BOOK_DIR:
                updatedRows = db.update("Book", values, selection, selectionArgs);
                break;
            case BOOK_ITEM:
                String bookId = uri.getPathSegments().get(1);
                updatedRows = db.update("Book", values, "id = ?", new String[]
                    { bookId });
                break;
            case CATEGORY_DIR:
                updatedRows = db.update("Category", values, selection,
                    selectionArgs);
                break;
            case CATEGORY_ITEM:
                String categoryId = uri.getPathSegments().get(1);
                updatedRows = db.update("Category", values, "id = ?", new String[]
                    { categoryId });
                break;
            default:
                break;
        }
        return updatedRows;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // 删除数据
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        int deletedRows = 0;
        switch (uriMatcher.match(uri)) {
            case BOOK_DIR:
                deletedRows = db.delete("Book", selection, selectionArgs);
                break;
            case BOOK_ITEM:
                String bookId = uri.getPathSegments().get(1);
                deletedRows = db.delete("Book", "id = ?", new String[] { bookId });
                break;
            case CATEGORY_DIR:
                deletedRows = db.delete("Category", selection, selectionArgs);
                break;
            case CATEGORY_ITEM:
```

```
                String categoryId = uri.getPathSegments().get(1);
                deletedRows = db.delete("Category", "id = ?", new String[]
                    { categoryId });
                break;
            default:
                break;
        }
        return deletedRows;
    }

    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)) {
            case BOOK_DIR:
                return "vnd.android.cursor.dir/vnd.com.example.databasetest.
                    provider.book";
            case BOOK_ITEM:
                return "vnd.android.cursor.item/vnd.com.example.databasetest.
                    provider.book";
            case CATEGORY_DIR:
                return "vnd.android.cursor.dir/vnd.com.example.databasetest.
                    provider.category";
            case CATEGORY_ITEM:
                return "vnd.android.cursor.item/vnd.com.example.databasetest.
                    provider.category";
        }
        return null;
    }

}
```

代码虽然很长，不过不用担心，这些内容都非常容易理解，因为使用到的全部都是上一小节中我们学到的知识。首先在类的一开始，同样是定义了 4 个常量，分别用于表示访问 Book 表中的所有数据、访问 Book 表中的单条数据、访问 Category 表中的所有数据和访问 Category 表中的单条数据。然后在静态代码块里对 UriMatcher 进行了初始化操作，将期望匹配的几种 URI 格式添加了进去。

接下来就是每个抽象方法的具体实现了，先来看下 `onCreate()`方法，这个方法的代码很短，就是创建了一个 MyDatabaseHelper 的实例，然后返回 `true` 表示内容提供器初始化成功，这时数据库就已经完成了创建或升级操作。

接着看一下 `query()`方法，在这个方法中先获取到了 SQLiteDatabase 的实例，然后根据传入的 `Uri` 参数判断出用户想要访问哪张表，再调用 SQLiteDatabase 的 `query()`进行查询，并将 `Cursor` 对象返回就好了。注意当访问单条数据的时候有一个细节，这里调用了 `Uri` 对象的 `getPathSegments()`方法，它会将内容 URI 权限之后的部分以“/”符号进行分割，并把分割后的结果放入到一个字符串列表中，那这个列表的第 0 个位置存放的就是路径，第 1 个位置存放的就是 id 了。得到了 id 之后，再通过 `selection` 和 `selectionArgs` 参数进行约束，就实现了查

询单条数据的功能。

再往后就是 insert()方法，同样它也是先获取到了 SQLiteDatabase 的实例，然后根据传入的 Uri 参数判断出用户想要往哪张表里添加数据，再调用 SQLiteDatabase 的 insert()方法进行添加就可以了。注意 insert()方法要求返回一个能够表示这条新增数据的 URI，所以我们还需要调用 Uri.parse()方法来将一个内容 URI 解析成 Uri 对象，当然这个内容 URI 是以新增数据的 id 结尾的。

接下来就是 update()方法了，相信这个方法中的代码已经完全难不倒你了。也是先获取 SQLiteDatabase 的实例，然后根据传入的 Uri 参数判断出用户想要更新哪张表里的数据，再调用 SQLiteDatabase 的 update()方法进行更新就好了，受影响的行数将作为返回值返回。

下面是 delete()方法，是不是感觉越到后面越轻松了？因为你已经渐入佳境，真正地找到窍门了。这里仍然是先获取到 SQLiteDatabase 的实例，然后根据传入的 Uri 参数判断出用户想要删除哪张表里的数据，再调用 SQLiteDatabase 的 delete()方法进行删除就好了，被删除的行数将作为返回值返回。

最后是 getType()方法，这个方法中的代码完全是按照上一节中介绍的格式规则编写的，相信已经没有什么解释的必要了。这样我们就将内容提供器中的代码全部编写完了。

另外还有一点需要注意，内容提供器一定要在 AndroidManifest.xml 文件中注册才可以使用。不过幸运的是，由于我们是使用 Android Studio 的快捷方式创建的内容提供器，因此注册这一步已经被自动完成了。打开 AndroidManifest.xml 文件瞧一瞧，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.databasetest">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <provider
            android:name=".DatabaseProvider"
            android:authorities="com.example.databasetest.provider"
            android:enabled="true"
            android:exported="true">
        </provider>
    </application>

</manifest>
```

可以看到，<application>标签内出现了一个新的标签<provider>，我们使用它来对 DatabaseProvider 这个内容提供器进行注册。android:name 属性指定了 DatabaseProvider 的类名，

android:authorities 属性指定了 DatabaseProvider 的 authority，而 enabled 和 exported 属性则是根据我们刚才勾选的状态自动生成的，这里表示允许 DatabaseProvider 被其他应用程序进行访问。

现在 DatabaseTest 这个项目就已经拥有了跨程序共享数据的功能了，我们赶快来尝试一下。首先需要将 DatabaseTest 程序从模拟器中删除掉，以防止上一章中产生的遗留数据对我们造成干扰。然后运行一下项目，将 DatabaseTest 程序重新安装在模拟器上了。接着关闭掉 DatabaseTest 这个项目，并创建一个新项目 ProviderTest，我们就将通过这个程序去访问 DatabaseTest 中的数据。

还是先来编写一下布局文件吧，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/add_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add To Book" />

    <Button
        android:id="@+id/query_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Query From Book" />

    <Button
        android:id="@+id/update_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Update Book" />

    <Button
        android:id="@+id/delete_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Delete From Book" />

</LinearLayout>
```

布局文件很简单，里面放置了 4 个按钮，分别用于添加、查询、修改和删除数据。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private String newId;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button addData = (Button) findViewById(R.id.add_data);
        addData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 添加数据
                Uri uri = Uri.parse("content://com.example.databasetest. provider/
                    book");
                ContentValues values = new ContentValues();
                values.put("name", "A Clash of Kings");
                values.put("author", "George Martin");
                values.put("pages", 1040);
                values.put("price", 22.85);
                Uri newUri = getContentResolver().insert(uri, values);
                newId = newUri.getPathSegments().get(1);
            }
        });
        Button queryData = (Button) findViewById(R.id.query_data);
        queryData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 查询数据
                Uri uri = Uri.parse("content://com.example.databasetest. provider/
                    book");
                Cursor cursor = getContentResolver().query(uri, null, null, null,
                    null);
                if (cursor != null) {
                    while (cursor.moveToNext()) {
                        String name = cursor.getString(cursor. getColumnIndex
                            ("name"));
                        String author = cursor.getString(cursor. getColumnIndex
                            ("author"));
                        int pages = cursor.getInt(cursor.getColumnIndex ("pages"));
                        double price = cursor.getDouble(cursor. getColumnIndex
                            ("price"));
                        Log.d("MainActivity", "book name is " + name);
                        Log.d("MainActivity", "book author is " + author);
                        Log.d("MainActivity", "book pages is " + pages);
                        Log.d("MainActivity", "book price is " + price);
                    }
                    cursor.close();
                }
            }
        });
        Button updateData = (Button) findViewById(R.id.update_data);
        updateData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 更新数据
                Uri uri = Uri.parse("content://com.example.databasetest. provider/
                    book/" + newId);
                ContentValues values = new ContentValues();
                values.put("name", "A Storm of Swords");
```

```
                values.put("pages", 1216);
                values.put("price", 24.05);
                getContentResolver().update(uri, values, null, null);
            }
        });
        Button deleteData = (Button) findViewById(R.id.delete_data);
        deleteData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 删除数据
                Uri uri = Uri.parse("content://com.example.databasetest. provider/
                    book/" + newId);
                getContentResolver().delete(uri, null, null);
            }
        });
    }

}
```

可以看到，我们分别在这 4 个按钮的点击事件里面处理了增删改查的逻辑。添加数据的时候，首先调用了 `Uri.parse()`方法将一个内容 URI 解析成 `Uri` 对象，然后把要添加的数据都存放到 `ContentValues` 对象中，接着调用 `ContentResolver` 的 `insert()`方法执行添加操作就可以了。注意 `insert()`方法会返回一个 `Uri` 对象，这个对象中包含了新增数据的 id，我们通过 `getPathSegments()`方法将这个 id 取出，稍后会用到它。

查询数据的时候，同样是调用了 `Uri.parse()`方法将一个内容 URI 解析成 `Uri` 对象，然后调用 `ContentResolver` 的 `query()`方法去查询数据，查询的结果当然还是存放在 `Cursor` 对象中的。之后对 `Cursor` 进行遍历，从中取出查询结果，并一一打印出来。

更新数据的时候，也是先将内容 URI 解析成 `Uri` 对象，然后把想要更新的数据存放到 `ContentValues` 对象中，再调用 `ContentResolver` 的 `update()`方法执行更新操作就可以了。注意这里我们为了不想让 Book 表中的其他行受到影响，在调用 `Uri.parse()`方法时，给内容 URI 的尾部增加了一个 id，而这个 id 正是添加数据时所返回的。这就表示我们只希望更新刚刚添加的那条数据，Book 表中的其他行都不会受影响。

删除数据的时候，也是使用同样的方法解析了一个以 id 结尾的内容 URI，然后调用 `ContentResolver` 的 `delete()`方法执行删除操作就可以了。由于我们在内容 URI 里指定了一个 id，因此只会删掉拥有相应 id 的那行数据，Book 表中的其他数据都不会受影响。

现在运行一下 ProviderTest 项目，会显示如图 7.14 所示的界面。

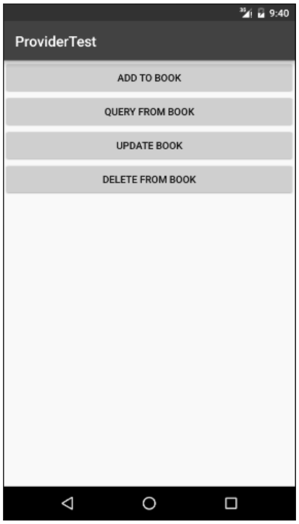


图 7.14　ProviderTest 主界面

点击一下 Add To Book 按钮，此时数据就应该已经添加到 DatabaseTest 程序的数据库中了，我们可以通过点击 Query From Book 按钮来检查一下，打印日志如图 7.15 所示。

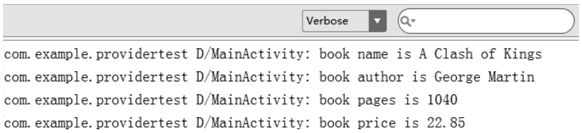


图 7.15　查询添加的数据

然后点击一下 Update Book 按钮来更新数据，再点击一下 Query From Book 按钮进行检查，结果如图 7.16 所示。

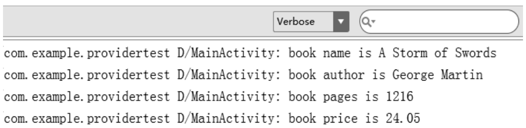


图 7.16　查询更新后的数据

最后点击 Delete From Book 按钮删除数据，此时再点击 Query From Book 按钮就查询不到数据了。由此可以看出，我们的跨程序共享数据功能已经成功实现了！现在不仅是 ProviderTest 程序，任何一个程序都可以轻松访问 DatabaseTest 中的数据，而且我们还丝毫不用担心隐私数据泄漏的问题。

到这里，与内容提供器相关的重要内容就基本全部介绍完了，下面就让我们再次进入本书的特殊环节，学习更多关于 Git 的用法。

7.5　Git 时间——版本控制工具进阶

在上一次的 Git 时间里，我们学习了关于 Git 最基本的用法，包括安装 Git、创建代码仓库，以及提交本地代码。本节中我们将要学习更多的使用技巧，不过在开始之前先要把准备工作做好。

所谓的准备工作就是要给一个项目创建代码仓库，这里就选择在 ProviderTest 项目中创建吧，打开 Git Bash，进入到这个项目的根目录下面，然后执行 `git init` 命令，如图 7.17 所示。

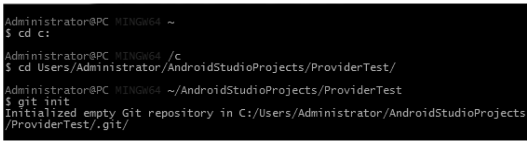


图 7.17　创建代码仓库

这样准备工作就已经完成了，让我们继续开始 Git 之旅吧。

7.5.1　忽略文件

代码仓库现在已经创建好了，接下来我们应该去提交 ProviderTest 项目中的代码。不过在提交之前你也许应该思考一下，是不是所有的文件都需要加入到版本控制当中呢？

在第 1 章介绍 Android 项目结构的时候有提到过，build 目录下的文件都是编译项目时自动生成的，我们不应该将这部分文件添加到版本控制当中，那么如何才能实现这样的效果呢？

Git 提供了一种可配性很强的机制来允许用户将指定的文件或目录排除在版本控制之外，它会检查代码仓库的目录下是否存在一个名为.gitignore 的文件，如果存在的话，就去一行行读取这个文件中的内容，并把每一行指定的文件或目录排除在版本控制之外。注意.gitignore 中指定的文件或目录是可以使用“*”通配符的。

神奇的是，我们并不需要自己去创建.gitignore 文件，Android Studio 在创建项目的时候会自动帮我们创建出两个.gitignore 文件，一个在根目录下面，一个在 app 模块下面。首先看一下根目录下面的.gitignore 文件，如图 7.18 所示。



图 7.18　根目录下面的.gitignore 文件

这是 Android Studio 自动生成的一些默认配置，通常情况下，这部分内容都是不用添加到版

本控制当中的。我们来简单阅读一下这个文件，除了*.iml 表示指定任意以.iml 结尾的文件，其他都是指定的具体的文件名或者目录名，上面配置中的所有内容都不会被添加到版本控制当中，因为基本都是一些由 IDE 自动生成的配置。

再来看一下 app 模块下面的.gitignore 文件，这个就简单多了，如图 7.19 所示。



图 7.19　app 模块下面的.gitignore 文件

由于 app 模块下面基本都是我们编写的代码，因此默认情况下只有其中的 build 目录不会被添加到版本控制当中。

当然，我们完全可以对以上两个文件进行任意地修改，来满足特定的需求。比如说，app 模块下面的所有测试文件都只是给我自己使用的，我并不想把它们添加到版本控制中，那么就可以这样修改 app/.gitignore 文件中的内容：

```
/build
/src/test
/src/androidTest
```

没错，只需添加这样两行配置，因为所有的测试文件都是放在这两个目录下的。现在我们可以提交代码了，先使用 add 命令将所有的文件进行添加，如下所示：

```
git add .
```

然后执行 commit 命令完成提交，如下所示：

```
git commit -m "First commit."
```

7.5.2　查看修改内容

在进行了第一次代码提交之后，我们后面还可能会对项目不断地进行维护或添加新功能等。比较理想的情况是每当完成了一小块功能，就执行一次提交。但是如果某个功能牵扯到的代码比较多，有可能写到后面的时候我们就已经忘记前面修改了什么东西了。遇到这种情况时不用担心，Git 全都帮你记着呢！下面我们就来学习一下如何使用 Git 来查看自上次提交后文件修改的内容。

查看文件修改情况的方法非常简单，只需要使用 status 命令就可以了，在项目的根目录下输入如下命令：

```
git status
```

然后 Git 会提示目前项目中没有任何可提交的文件，因为我们刚刚才提交过嘛。现在对 ProviderTest 项目中的代码稍做一下改动，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        addData.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                ...
                values.put("price", 55.55);
                ...
            }
        });
        ...
    }
}
```

这里仅仅是在添加数据的时候，将书的价格由 22.85 改成了 55.55。然后重新输入 `git status` 命令，这次结果如图 7.20 所示。

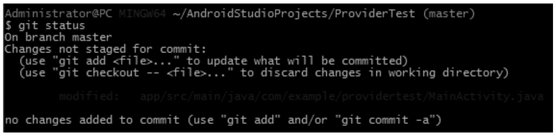


图 7.20 查看文件变动情况

可以看到，Git 提醒我们 MainActivity.java 这个文件已经发生了更改，那么如何才能看到更改的内容呢？这就需要借助 `diff` 命令了，用法如下所示：

```
git diff
```

这样可以查看到所有文件的更改内容，如果你只想查看 MainActivity.java 这个文件的更改内容，可以使用如下命令：

```
git diff app/src/main/java/com/example/providertest/MainActivity.java
```

命令的执行结果如图 7.21 所示。



图 7.21 查看修改的具体内容

其中，减号代表删除的部分，加号代表添加的部分。从图中我们就可以明显地看出，书的价格由 22.85 被修改成了 55.55。

7.5.3　撤销未提交的修改

有时候我们的代码可能会写得过于草率，以至于原本正常的功能，结果反倒被我们改出了问题。遇到这种情况时也不用着急，因为只要代码还未提交，所有修改的内容都是可以撤销的。

比如在上一小节中我们修改了 MainActivity 里一本书的价格，现在如果想要撤销这个修改就可以使用 `checkout` 命令，用法如下所示：

```
git checkout app/src/main/java/com/example/providertest/MainActivity.java
```

执行了这个命令之后，我们对 MainActivity.java 这个文件所做的一切修改就应该都被撤销了。重新运行 `git status` 命令检查一下，结果如图 7.22 所示。

```
Administrator@PC MINGW64 ~/AndroidStudioProjects/ProviderTest (master)
$ git status
On branch master
nothing to commit, working directory clean
```

图 7.22　重新查看文件变动情况

可以看到，当前项目中没有任何可提交的文件，说明撤销操作确实是成功了。

不过这种撤销方式只适用于那些还没有执行过 `add` 命令的文件，如果某个文件已经被添加过了，这种方式就无法撤销其更改的内容，我们来做个试验瞧一瞧。

首先仍然是将 MainActivity 中那本书的价格改成 55.55，然后输入如下命令：

```
git add .
```

这样就把所有修改的文件都进行了添加，可以输入 `git status` 来检查一下，结果如图 7.23 所示。

```
Administrator@PC MINGW64 ~/AndroidStudioProjects/ProviderTest (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   app/src/main/java/com/example/providertest/MainActivity.java
```

图 7.23　再次查看文件变动情况

现在我们再执行一遍 `checkout` 命令，你会发现 MainActivity 仍然是处于已添加状态，所修改的内容无法撤销掉。

这种情况应该怎么办？难道我们还没法后悔了？当然不是，只不过对于已添加的文件我们应该先对其取消添加，然后才可以撤回提交。取消添加使用的是 `reset` 命令，用法如下所示：

```
git reset HEAD app/src/main/java/com/example/providertest/MainActivity.java
```

然后再运行一遍 `git status` 命令，你就会发现 MainActivity.java 这个文件重新变回了未添加状态，此时就可以使用 `checkout` 命令来将修改的内容进行撤销了。

7.5.4 查看提交记录

当 ProviderTest 这个项目开发了几个月之后，我们可能已经执行过上百次的提交操作了，这个时候估计你早就已经忘记每次提交都修改了哪些内容。不过没关系，忠实的 Git 一直都帮我们清清楚楚地记录着呢！可以使用 `log` 命令查看历史提交信息，用法如下所示：

```
git log
```

由于目前我们只执行过一次提交，所以能看到的信息很少，如图 7.24 所示。

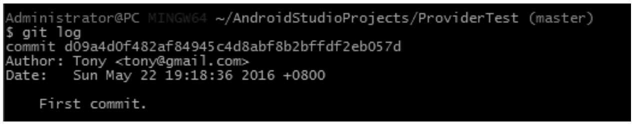


图 7.24 查看提交记录

可以看到，每次提交记录都会包含提交 id、提交人、提交日期以及提交描述这 4 个信息。那么我们再次将书价修改成 55.55，然后执行一次提交操作，如下所示：

```
git add .
git commit -m "Change price."
```

现在重新执行 `git log` 命令，结果如图 7.25 所示。

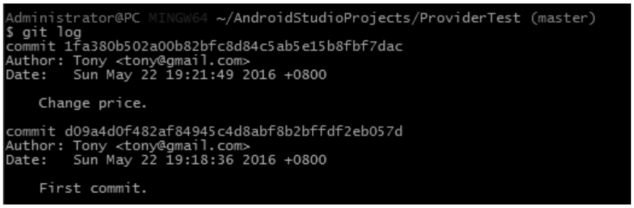


图 7.25 重新查看提交记录

当提交记录非常多的时候，如果我们只想查看其中一条记录，可以在命令中指定该记录的 id，并加上 `-1` 参数表示我们只想看到一行记录，如下所示：

```
git log 1fa380b502a00b82bfc8d84c5ab5e15b8fbf7dac -1
```

而如果想要查看这条提交记录具体修改了什么内容，可以在命令中加入 `-p` 参数，命令如下：

```
git log 1fa380b502a00b82bfc8d84c5ab5e15b8fbf7dac -1 –p
```

查询出的结果如图 7.26 所示，其中减号代表删除的部分，加号代表添加的部分。

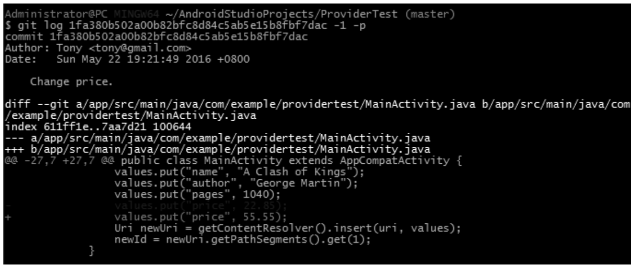


图 7.26　查看提交记录的具体修改内容

好了，本次的 Git 时间就到这里，下面我们来对本章中所学的知识做个回顾吧。

7.6　小结与点评

本章的内容不算多，而且很多时候都是在使用上一章中学习的数据库知识，所以理解这部分内容对你来说应该是比较轻松的吧。在本章中，我们一开始先了解了 Android 的权限机制，并且学会了如何在 6.0 以上的系统中使用运行时权限，然后又重点学习了内容提供器的相关内容，以实现跨程序数据共享的功能。现在你不仅知道了如何去访问其他程序中的数据，还学会了怎样创建自己的内容提供器来共享数据，收获还是挺大的吧。

不过每次在创建内容提供器的时候，你都需要提醒一下自己，我是不是应该这么做？因为只有真正需要将数据共享出去的时候我们才应该创建内容提供器，仅仅是用于程序内部访问的数据就没有必要这么做，所以千万别对它进行滥用。

在连续学了几章系统机制方面的内容之后是不是感觉有些枯燥？那么下一章中我们就来换换口味，学习一下 Android 多媒体方面的知识吧。

第 8 章

丰富你的程序——运用手机多媒体

在过去，手机的功能都比较单调，仅仅就是用来打电话和发短信的。而如今，手机在我们的生活中正扮演着越来越重要的角色，各种娱乐方式都可以在手机上进行。上班的路上太无聊，可以戴着耳机听音乐。外出旅行的时候，可以在手机上看电影。无论走到哪里，遇到喜欢的事物都可以随手拍下来。

众多的娱乐方式少不了强大的多媒体功能的支持，而 Android 在这方面也做得非常出色。它提供了一系列的 API，使得我们可以在程序中调用很多手机的多媒体资源，从而编写出更加丰富多彩的应用程序，本章我们就将对 Android 中一些常用的多媒体功能的使用技巧进行学习。

前面的 7 章内容，我们一直都是使用模拟器来运行程序的，不过本章涉及的一些功能必须要在真正的 Android 手机上运行才看得到效果。因此，首先我们就来学习一下，如何使用 Android 手机来运行程序。

8.1 将程序运行到手机上

不必我多说，首先你需要拥有一部 Android 手机。现在 Android 手机早就不是什么稀罕物，几乎已经是人手一部了，如果你还没有的话，赶紧去购买吧。

想要将程序运行到手机上，我们需要先通过数据线把手机连接到电脑上。然后进入到设置→开发者选项界面，并在这个界面中勾选中 USB 调试选项，如图 8.1 所示。



图 8.1　启用 USB 调试

注意从 Android 4.2 系统开始，开发者选项默认是隐藏的，你需要先进入到“关于手机”界面，然后对着最下面的版本号那一栏连续点击，就会让开发者选项显示出来。

然后如果你使用的是 Windows 操作系统，还需要在电脑上安装手机的驱动。一般借助 360 手机助手或豌豆荚等工具都可以快速地进行安装，安装完成后就可以看到手机已经连接到电脑上了，如图 8.2 所示。



图 8.2　手机成功连接上电脑

现在观察 Android Monitor，你会发现当前是有两个设备在线的，一个是我们一直使用的模拟器，另外一个则是刚刚连接上的手机了，如图 8.3 所示。

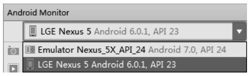


图 8.3　在线设备列表

然后运行一下当前项目，这时不会直接将程序运行到模拟器或者手机上，而是会弹出一个对话框让你进行选择，如图 8.4 所示。

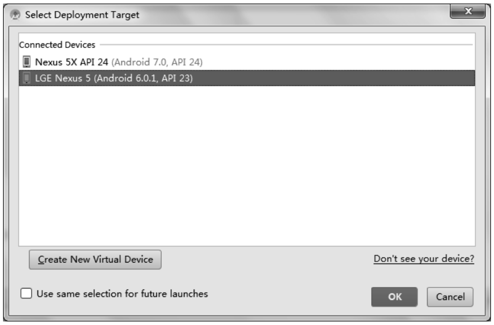


图 8.4　选择运行设备对话框

选中下面的 LGE Nexus 5 后点击 OK，就会将程序运行到手机上了。

8.2　使用通知

通知（Notification）是 Android 系统中比较有特色的一个功能，当某个应用程序希望向用户发出一些提示信息，而该应用程序又不在前台运行时，就可以借助通知来实现。发出一条通知后，手机最上方的状态栏中会显示一个通知的图标，下拉状态栏后可以看到通知的详细内容。Android 的通知功能获得了大量用户的认可和喜爱，就连 iOS 系统也在 5.0 版本之后加入了类似的功能。

8.2.1　通知的基本用法

了解了通知的基本概念，下面我们就来看一下通知的使用方法吧。通知的用法还是比较灵活的，既可以在活动里创建，也可以在广播接收器里创建，当然还可以在下一章中我们即将学习的服务里创建。相比于广播接收器和服务，在活动里创建通知的场景还是比较少的，因为一般只有当程序进入到后台的时候我们才需要使用通知。

不过，无论是在哪里创建通知，整体的步骤都是相同的，下面我们就来学习一下创建通知的详细步骤。首先需要一个 NotificationManager 来对通知进行管理，可以调用 Context 的 `getSystemService()` 方法获取到。`getSystemService()` 方法接收一个字符串参数用于确定获取系统的哪个服务，这里我们传入 `Context.NOTIFICATION_SERVICE` 即可。因此，获取 NotificationManager 的实例就可以写成：

```
NotificationManager manager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
```

接下来需要使用一个 Builder 构造器来创建 Notification 对象，但问题在于，几乎 Android 系统的每一个版本都会对通知这部分功能进行或多或少的修改，API 不稳定性问题在通知上面突显得尤其严重。那么该如何解决这个问题呢？其实解决方案我们之前已经见过好几回了，就是使用 support 库中提供的兼容 API。support-v4 库中提供了一个 NotificationCompat 类，使用这个类的构造器来创建 Notification 对象，就可以保证我们的程序在所有 Android 系统版本上都能正常工作了，代码如下所示：

```
Notification notification = new NotificationCompat.Builder(context).build();
```

当然，上述代码只是创建了一个空的 Notification 对象，并没有什么实际作用，我们可以在最终的 build()方法之前连缀任意多的设置方法来创建一个丰富的 Notification 对象，先来看一些最基本的设置：

```
Notification notification = new NotificationCompat.Builder(context)
        .setContentTitle("This is content title")
        .setContentText("This is content text")
        .setWhen(System.currentTimeMillis())
        .setSmallIcon(R.drawable.small_icon)
        .setLargeIcon(BitmapFactory.decodeResource(getResources(),
            R.drawable.large_icon))
        .build();
```

上述代码中一共调用了 5 个设置方法，下面我们来一一解析一下。setContentTitle()方法用于指定通知的标题内容，下拉系统状态栏就可以看到这部分内容。setContentText()方法用于指定通知的正文内容，同样下拉系统状态栏就可以看到这部分内容。setWhen()方法用于指定通知被创建的时间，以毫秒为单位，当下拉系统状态栏时，这里指定的时间会显示在相应的通知上。setSmallIcon()方法用于设置通知的小图标，注意只能使用纯 alpha 图层的图片进行设置，小图标会显示在系统状态栏上。setLargeIcon()方法用于设置通知的大图标，当下拉系统状态栏时，就可以看到设置的大图标了。

以上工作都完成之后，只需要调用 NotificationManager 的 notify()方法就可以让通知显示出来了。notify()方法接收两个参数，第一个参数是 id，要保证为每个通知所指定的 id 都是不同的。第二个参数则是 Notification 对象，这里直接将我们刚刚创建好的 Notification 对象传入即可。因此，显示一个通知就可以写成：

```
manager.notify(1, notification);
```

到这里就已经把创建通知的每一个步骤都分析完了，下面就让我们通过一个具体的例子来看一看通知到底是长什么样的。

新建一个 NotificationTest 项目，并修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/send_notice"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send notice" />

</LinearLayout>
```

布局文件非常简单，里面只有一个 Send notice 按钮，用于发出一条通知。接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button sendNotice = (Button) findViewById(R.id.send_notice);
        sendNotice.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.send_notice:
                NotificationManager manager = (NotificationManager) getSystemService
                    (NOTIFICATION_SERVICE);
                Notification notification = new NotificationCompat.Builder(this)
                        .setContentTitle("This is content title")
                        .setContentText("This is content text")
                        .setWhen(System.currentTimeMillis())
                        .setSmallIcon(R.mipmap.ic_launcher)
                        .setLargeIcon(BitmapFactory.decodeResource(getResources(),
                            R.mipmap.ic_launcher))
                        .build();
                manager.notify(1, notification);
                break;
            default:
                break;
        }
    }

}
```

可以看到，我们在 Send notice 按钮的点击事件里面完成了通知的创建工作，创建的过程正如前面所描述的一样。不过这里简单起见，我将通知栏的大小图都直接设置成了 ic_launcher 这张图，

这样就不用再去专门准备图标了，而在实际项目中千万不要这样偷懒。

现在可以来运行一下程序了，点击 Send notice 按钮，你会在系统状态栏的最左边看到一个小图标，如图 8.5 所示。

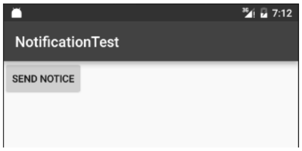


图 8.5　通知的小图标

下拉系统状态栏可以看到该通知的详细信息，如图 8.6 所示。

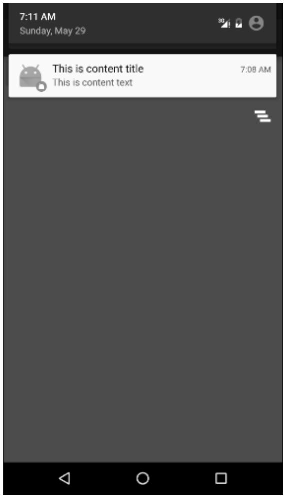


图 8.6　通知的详细信息

如果你使用过 Android 手机，此时应该会下意识地认为这条通知是可以点击的。但是当你去点击它的时候，你会发现没有任何效果。不对啊，好像每条通知点击之后都应该会有反应的呀？其实要想实现通知的点击效果，我们还需要在代码中进行相应的设置，这就涉及了一个新的概念：PendingIntent。

PendingIntent 从名字上看起来就和 Intent 有些类似，它们之间也确实存在着不少共同点。比如它们都可以去指明某一个“意图”，都可以用于启动活动、启动服务以及发送广播等。不同的是，Intent 更加倾向于去立即执行某个动作，而 PendingIntent 更加倾向于在某个合适的时机去执行某个动作。所以，也可以把 PendingIntent 简单地理解为延迟执行的 Intent。

PendingIntent 的用法同样很简单，它主要提供了几个静态方法用于获取 PendingIntent 的实例，可以根据需求来选择是使用 `getActivity()`方法、`getBroadcast()`方法，还是 `getService()`

方法。这几个方法所接收的参数都是相同的，第一个参数依旧是 Context，不用多做解释。第二个参数一般用不到，通常都是传入 0 即可。第三个参数是一个 Intent 对象，我们可以通过这个对象构建出 PendingIntent 的“意图”。第四个参数用于确定 PendingIntent 的行为，有 FLAG_ONE_SHOT、FLAG_NO_CREATE、FLAG_CANCEL_CURRENT 和 FLAG_UPDATE_CURRENT 这 4 种值可选，每种值的具体含义你可以查看文档，通常情况下这个参数传入 0 就可以了。

对 PendingIntent 有了一定的了解后，我们再回过头来看一下 NotificationCompat.Builder。这个构造器还可以再连缀一个 setContentIntent()方法，接收的参数正是一个 PendingIntent 对象。因此，这里就可以通过 PendingIntent 构建出一个延迟执行的“意图”，当用户点击这条通知时就会执行相应的逻辑。

现在我们来优化一下 NotificationTest 项目，给刚才的通知加上点击功能，让用户点击它的时候可以启动另一个活动。

首先需要准备好另一个活动，右击 com.example.notificationtest 包→New→Activity→Empty Activity，新建 NotificationActivity，布局起名为 notification_layout。然后修改 notification_layout.xml 中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:textSize="24sp"
        android:text="This is notification layout"
        />

</RelativeLayout>
```

这样就把 NotificationActivity 这个活动准备好了，下面我们修改 MainActivity 中的代码，给通知加入点击功能，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.send_notice:
                Intent intent = new Intent(this, NotificationActivity.class);
                PendingIntent pi = PendingIntent.getActivity(this, 0, intent, 0);
                NotificationManager manager = (NotificationManager) getSystemService
                    (NOTIFICATION_SERVICE);
                Notification notification = new NotificationCompat.Builder(this)
                        .setContentTitle("This is content title")
```

```
                    .setContentText("This is content text")
                    .setWhen(System.currentTimeMillis())
                    .setSmallIcon(R.mipmap.ic_launcher)
                    .setLargeIcon(BitmapFactory.decodeResource(getResources(),
                        R.mipmap.ic_launcher))
                    .setContentIntent(pi)
                    .build();
                manager.notify(1, notification);
                break;
            default:
                break;
        }
    }

}
```

可以看到，这里先是使用 Intent 表达出我们想要启动 NotificationActivity 的“意图”，然后将构建好的 Intent 对象传入到 PendingIntent 的 getActivity()方法里，以得到 PendingIntent 的实例，接着在 NotificationCompat.Builder 中调用 setContentIntent()方法，把它作为参数传入即可。

现在重新运行一下程序，并点击 Send notice 按钮，依旧会发出一条通知。然后下拉系统状态栏，点击一下该通知，就会看到 NotificationActivity 这个活动的界面了，如图 8.7 所示。

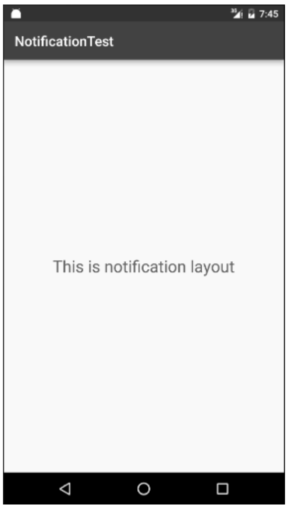


图 8.7　点击通知后打开 NotificationActivity 界面

咦？怎么系统状态上的通知图标还没有消失呢？是这样的，如果我们没有在代码中对该通知进行取消，它就会一直显示在系统的状态栏上。解决的方法有两种，一种是在 NotificationCompat.Builder 中再连缀一个 setAutoCancel()方法，一种是显式地调用 NotificationManager 的 cancel()方法将它取消，两种方法我们都学习一下。

第一种方法写法如下：

```
Notification notification = new NotificationCompat.Builder(this)
        ...
        .setAutoCancel(true)
        .build();
```

可以看到，setAutoCancel()方法传入 true，就表示当点击了这个通知的时候，通知会自动取消掉。

第二种方法写法如下：

```
public class NotificationActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notification_layout);
        NotificationManager manager = (NotificationManager) getSystemService
           (NOTIFICATION_SERVICE);
        manager.cancel(1);
    }
}
```

这里我们在 cancel()方法中传入了 1，这个 1 是什么意思呢？还记得在创建通知的时候给每条通知指定的 id 吗？当时我们给这条通知设置的 id 就是 1。因此，如果你想取消哪条通知，在 cancel()方法中传入该通知的 id 就行了。

8.2.2 通知的进阶技巧

现在你已经掌握了创建和取消通知的方法，并且知道了如何去响应通知的点击事件。不过通知的用法并不仅仅是这些呢，下面我们就来探究一下通知的更多技巧。

上一小节中创建的通知属于最基本的通知，实际上，NotificationCompat.Builder 中提供了非常丰富的 API 来让我们创建出更加多样的通知效果。当然，每一个 API 都详细地讲一遍不太可能，我们只能从中选一些比较常用的 API 来进行学习。先来看看 setSound()方法吧，它可以在通知发出的时候播放一段音频，这样就能够更好地告知用户有通知到来。setSound()方法接收一个 Uri 参数，所以在指定音频文件的时候还需要先获取到音频文件对应的 URI。比如说，每个手机的/system/media/audio/ringtones 目录下都有很多的音频文件，我们可以从中随便选一个音频文件，那么在代码中就可以这样指定：

```
Notification notification = new NotificationCompat.Builder(this)
        ...
        .setSound(Uri.fromFile(new File("/system/media/audio/ringtones/Luna.ogg")))
        .build();
```

除了允许播放音频外，我们还可以在通知到来的时候让手机进行振动，使用的是 vibrate

这个属性。它是一个长整型的数组，用于设置手机静止和振动的时长，以毫秒为单位。下标为 0 的值表示手机静止的时长，下标为 1 的值表示手机振动的时长，下标为 2 的值又表示手机静止的时长，以此类推。所以，如果想要让手机在通知到来的时候立刻振动 1 秒，然后静止 1 秒，再振动 1 秒，代码就可以写成：

```
Notification notification = new NotificationCompat.Builder(this)
        ...
        .setVibrate(new long[] {0, 1000, 1000, 1000 })
        .build();
```

不过，想要控制手机振动还需要声明权限。因此，我们还得编辑 AndroidManifest.xml 文件，加入如下声明：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.notificationtest"
    android:versionCode="1"
    android:versionName="1.0" >
    ...
    <uses-permission android:name="android.permission.VIBRATE" />
    ...
</manifest>
```

学会了控制通知的声音和振动，下面我们来看一下如何在通知到来时控制手机 LED 灯的显示。

现在的手机基本上都会前置一个 LED 灯，当有未接电话或未读短信，而此时手机又处于锁屏状态时，LED 灯就会不停地闪烁，提醒用户去查看。我们可以使用 `setLights()`方法来实现这种效果，`setLights()`方法接收 3 个参数，第一个参数用于指定 LED 灯的颜色，第二个参数用于指定 LED 灯亮起的时长，以毫秒为单位，第三个参数用于指定 LED 灯暗去的时长，也是以毫秒为单位。所以，当通知到来时，如果想要实现 LED 灯以绿色的灯光一闪一闪的效果，就可以写成：

```
Notification notification = new NotificationCompat.Builder(this)
        ...
        .setLights(Color.GREEN, 1000, 1000)
        .build();
```

当然，如果你不想进行那么多繁杂的设置，也可以直接使用通知的默认效果，它会根据当前手机的环境来决定播放什么铃声，以及如何振动，写法如下：

```
Notification notification = new NotificationCompat.Builder(this)
        ...
        .setDefaults(NotificationCompat.DEFAULT_ALL)
        .build();
```

注意，以上所涉及的这些进阶技巧都要在手机上运行才能看得到效果，模拟器是无法表现出振动以及 LED 灯闪烁等功能的。

8.2.3　通知的高级功能

继续观察 NotificationCompat.Builder 这个类，你会发现里面还有很多 API 是我们没有使用过的。那么下面我们就来学习一些更加强大的 API 的用法，从而构建出更加丰富的通知效果。

先来看看 setStyle()方法，这个方法允许我们构建出富文本的通知内容。也就是说通知中不光可以有文字和图标，还可以包含更多的东西。setStyle()方法接收一个 NotificationCompat.Style 参数，这个参数就是用来构建具体的富文本信息的，如长文字、图片等。

在开始使用 setStyle()方法之前，我们先来做一个试验吧，之前的通知内容都比较短，如果设置成很长的文字会是什么效果呢？比如这样写：

```
Notification notification = new NotificationCompat.Builder(this)
        ...
        .setContentText("Learn how to build notifications, send and sync data, and use
            voice actions. Get the official Android IDE and developer tools to build
            apps for Android.")
        ...
        .build();
```

现在重新运行程序并触发通知，效果如图 8.8 所示。

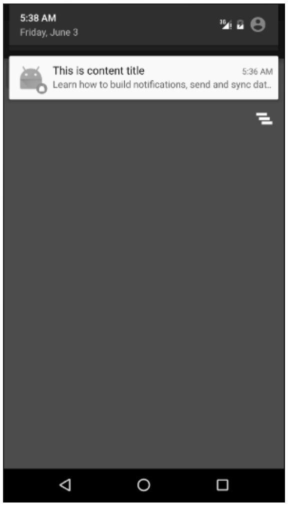


图 8.8　通知内容文字过长的效果

可以看到，通知内容是无法显示完整的，多余的部分会用省略号来代替。其实这也很正常，因为通知的内容本来就应该言简意赅，详细内容放到点击后打开的活动当中会更加合适。

但是如果你真的非常需要在通知当中显示一段长文字，Android 也是支持的，通过 setStyle()方法就可以做到，具体写法如下：

```
Notification notification = new NotificationCompat.Builder(this)
```

```
        ...
        .setStyle(new NotificationCompat.BigTextStyle().bigText("Learn how to build
            notifications, send and sync data, and use voice actions. Get the official
            Android IDE and developer tools to build apps for Android."))
        .build();
```

我们在 setStyle()方法中创建了一个 NotificationCompat.BigTextStyle 对象，这个对象就是用于封装长文字信息的，我们调用它的 bigText()方法并将文字内容传入就可以了。

再次重新运行程序并触发通知，效果如图 8.9 所示。

除了显示长文字之外，通知里还可以显示一张大图片，具体用法也是基本相似的：

```
Notification notification = new NotificationCompat.Builder(this)
        ...
        .setStyle(new NotificationCompat.BigPictureStyle().bigPicture
            (BitmapFactory.decodeResource(getResources(), R.drawable.big_image)))
        .build();
```

可以看到，这里仍然是调用的 setStyle()方法，这次我们在参数中创建了一个 NotificationCompat.BigPictureStyle 对象，这个对象就是用于设置大图片的，然后调用它的 bigPicture()方法并将图片传入。这里我事先准备好了一张图片，通过 BitmapFactory 的 decodeResource()方法将图片解析成 Bitmap 对象，再传入到 bigPicture()方法中就可以了。

现在重新运行一下程序并触发通知，效果如图 8.10 所示。



图 8.9　通知中显示长文字的效果



图 8.10　通知中显示大图片的效果

这样我们就把 setStyle()方法中的重要内容基本都掌握了。

接下来再学习一下 setPriority()方法，它可以用于设置通知的重要程度。setPriority()

方法接收一个整型参数用于设置这条通知的重要程度，一共有 5 个常量值可选：PRIORITY_DEFAULT 表示默认的重要程度，和不设置效果是一样的；PRIORITY_MIN 表示最低的重要程度，系统可能只会在特定的场景才显示这条通知，比如用户下拉状态栏的时候；PRIORITY_LOW 表示较低的重要程度，系统可能会将这类通知缩小，或改变其显示的顺序，将其排在更重要的通知之后；PRIORITY_HIGH 表示较高的重要程度，系统可能会将这类通知放大，或改变其显示的顺序，将其排在比较靠前的位置；PRIORITY_MAX 表示最高的重要程度，这类通知消息必须要让用户立刻看到，甚至需要用户做出响应操作。具体写法如下：

```
Notification notification = new NotificationCompat.Builder(this)
        ...
        .setPriority(NotificationCompat.PRIORITY_MAX)
        .build();
```

这里我们将通知的重要程度设置成了最高，表示这是一条非常重要的通知，要求用户必须立刻看到。现在重新运行一下程序，并点击 Send notice 按钮，效果如图 8.11 所示。



图 8.11　触发一条重要通知

可以看到，这次的通知不是在系统状态栏显示一个小图标了，而是弹出了一个横幅，并附带了通知的详细内容，表示这是一条非常重要的通知。不管用户现在是在玩游戏还是看电影，这条通知都会显示在最上方，以此引起用户的注意。当然，使用这类通知时一定要小心，确保你的通知内容的确是至关重要的，不然如果让用户产生反感的话，很可能会导致我们的应用程序被卸载。

8.3　调用摄像头和相册

我们平时在使用 QQ 或微信的时候经常要和别人分享图片，这些图片可以是用手机摄像头拍

的，也可以是从相册中选取的。类似这样的功能实在是太常见了，几乎在每一个应用程序中都会有，那么本节我们就学习一下调用摄像头和相册方面的知识。

8.3.1　调用摄像头拍照

先来看看摄像头方面的知识，现在很多的应用都会要求用户上传一张图片来作为头像，这时打开摄像头拍张照是最简单快捷的。下面就让我们通过一个例子来学习一下，如何才能在应用程序里调用手机的摄像头进行拍照。

新建一个 CameraAlbumTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/take_photo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Take Photo" />

    <ImageView
        android:id="@+id/picture"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

可以看到，布局文件中只有两个控件，一个 Button 和一个 ImageView。Button 是用于打开摄像头进行拍照的，而 ImageView 则是用于将拍到的图片显示出来。

然后开始编写调用摄像头的具体逻辑，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    public static final int TAKE_PHOTO = 1;

    private ImageView picture;

    private Uri imageUri;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button takePhoto = (Button) findViewById(R.id.take_photo);
        picture = (ImageView) findViewById(R.id.picture);
        takePhoto.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
                public void onClick(View v) {
                    // 创建File对象，用于存储拍照后的图片
                    File outputImage = new File(getExternalCacheDir(),
                        "output_image.jpg");
                    try {
                        if (outputImage.exists()) {
                            outputImage.delete();
                        }
                        outputImage.createNewFile();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                    if (Build.VERSION.SDK_INT >= 24) {
                        imageUri = FileProvider.getUriForFile(MainActivity.this,
                            "com.example.cameraalbumtest.fileprovider", outputImage);
                    } else {
                        imageUri = Uri.fromFile(outputImage);
                    }
                    // 启动相机程序
                    Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
                    intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
                    startActivityForResult(intent, TAKE_PHOTO);
                }
            });
        }

        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            switch (requestCode) {
                case TAKE_PHOTO:
                    if (resultCode == RESULT_OK) {
                        try {
                            // 将拍摄的照片显示出来
                            Bitmap bitmap = BitmapFactory.decodeStream(getContent-
                                Resolver().openInputStream(imageUri));
                            picture.setImageBitmap(bitmap);
                        } catch (FileNotFoundException e) {
                            e.printStackTrace();
                        }
                    }
                    break;
                default:
                    break;
            }
        }

    }
```

上述代码稍微有点复杂，我们来仔细地分析一下。在 MainActivity 中要做的第一件事自然是分别获取到 Button 和 ImageView 的实例，并给 Button 注册上点击事件，然后在 Button 的点击事件里开始处理调用摄像头的逻辑，我们重点看一下这部分代码。

首先这里创建了一个 File 对象，用于存放摄像头拍下的图片，这里我们把图片命名为 output_image.jpg，并将它存放在手机 SD 卡的应用关联缓存目录下。什么叫作应用关联缓存目录呢？就是指 SD 卡中专门用于存放当前应用缓存数据的位置，调用 getExternalCacheDir()方法可以得到这个目录，具体的路径是/sdcard/Android/data/<package name>/cache。那么为什么要使用应用关联缓目录来存放图片呢？因为从 Android 6.0 系统开始，读写 SD 卡被列为了危险权限，如果将图片存放在 SD 卡的任何其他目录，都要进行运行时权限处理才行，而使用应用关联目录则可以跳过这一步。

接着会进行一个判断，如果运行设备的系统版本低于 Android 7.0，就调用 Uri 的 fromFile()方法将 File 对象转换成 Uri 对象，这个 Uri 对象标识着 output_image.jpg 这张图片的本地真实路径。否则，就调用 FileProvider 的 getUriForFile()方法将 File 对象转换成一个封装过的 Uri 对象。getUriForFile()方法接收 3 个参数，第一个参数要求传入 Context 对象，第二个参数可以是任意唯一的字符串，第三个参数则是我们刚刚创建的 File 对象。之所以要进行这样一层转换，是因为从 Android 7.0 系统开始，直接使用本地真实路径的 Uri 被认为是不安全的，会抛出一个 FileUriExposedException 异常。而 FileProvider 则是一种特殊的内容提供器，它使用了和内容提供器类似的机制来对数据进行保护，可以选择性地将封装过的 Uri 共享给外部，从而提高了应用的安全性。

接下来构建出了一个 Intent 对象，并将这个 Intent 的 action 指定为 android.media.action.IMAGE_CAPTURE，再调用 Intent 的 putExtra()方法指定图片的输出地址，这里填入刚刚得到的 Uri 对象，最后调用 startActivityForResult()来启动活动。由于我们使用的是一个隐式 Intent，系统会找出能够响应这个 Intent 的活动去启动，这样照相机程序就会被打开，拍下的照片将会输出到 output_image.jpg 中。

注意，刚才我们是使用 startActivityForResult()来启动活动的，因此拍完照后会有结果返回到 onActivityResult()方法中。如果发现拍照成功，就可以调用 BitmapFactory 的 decodeStream()方法将 output_image.jpg 这张照片解析成 Bitmap 对象，然后把它设置到 ImageView 中显示出来。

不过现在还没结束，刚才提到了内容提供器，那么我们自然要在 AndroidManifest.xml 中对内容提供器进行注册了，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.cameraalbumtest">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <provider
            android:name="android.support.v4.content.FileProvider"
```

```
                android:authorities="com.example.cameraalbumtest.fileprovider"
                android:exported="false"
                android:grantUriPermissions="true">
                <meta-data
                    android:name="android.support.FILE_PROVIDER_PATHS"
                    android:resource="@xml/file_paths" />
            </provider>
        </application>
    </manifest>
```

其中，android:name 属性的值是固定的，android:authorities 属性的值必须要和刚才 FileProvider.getUriForFile()方法中的第二个参数一致。另外，这里还在<provider>标签的内部使用<meta-data>来指定 Uri 的共享路径，并引用了一个@xml/file_paths 资源。当然，这个资源现在还是不存在的，下面我们就来创建它。

右击 res 目录→New→Directory，创建一个 xml 目录，接着右击 xml 目录→New→File，创建一个 file_paths.xml 文件。然后修改 file_paths.xml 文件中的内容，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="my_images" path="" />
</paths>
```

其中，external-path 就是用来指定 Uri 共享的，name 属性的值可以随便填，path 属性的值表示共享的具体路径。这里设置空值就表示将整个 SD 卡进行共享，当然你也可以仅共享我们存放 output_image.jpg 这张图片的路径。

另外还有一点要注意，在 Android 4.4 系统之前，访问 SD 卡的应用关联目录也是要声明权限的，从 4.4 系统开始不再需要权限声明。那么我们为了能够兼容老版本系统的手机，还需要在 AndroidManifest.xml 中声明一下访问 SD 卡的权限：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.cameraalbumtest">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

这样代码就都编写完了，现在将程序运行到手机上，然后点击 Take Photo 按钮就可以进行拍照了，如图 8.12 所示。拍照完成后，点击中间按钮就会回到我们程序的界面。同时，拍摄的照片也会显示出来了，如图 8.13 所示。

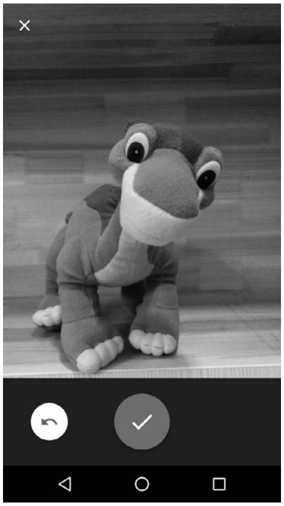


图 8.12　打开摄像头拍照



图 8.13　拍照的最终效果

8.3.2　从相册中选择照片

虽然调用摄像头拍照既方便又快捷，但我们并不是每次都需要去当场拍一张照片的。因为每个人的手机相册里应该都会存有许许多多张照片，直接从相册里选取一张现有的照片会比打开相机拍一张照片更加常用。一个优秀的应用程序应该将这两种选择方式都提供给用户，由用户来决定使用哪一种。下面我们就来看一下，如何才能实现从相册中选择照片的功能。

还是在 CameraAlbumTest 项目的基础上进行修改，编辑 activity_main.xml 文件，在布局中添加一个按钮用于从相册中选择照片，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/take_photo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Take Photo" />

    <Button
        android:id="@+id/choose_from_album"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Choose From Album" />

    <ImageView
        android:id="@+id/picture"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

然后修改 MainActivity 中的代码，加入从相册选择照片的逻辑，代码如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    public static final int CHOOSE_PHOTO = 2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button takePhoto = (Button) findViewById(R.id.take_photo);
        Button chooseFromAlbum = (Button) findViewById(R.id.choose_from_album);
        ...
        chooseFromAlbum.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (ContextCompat.checkSelfPermission(MainActivity.this,
                    Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.
                       PERMISSION_GRANTED) {
                    ActivityCompat.requestPermissions(MainActivity.this, new
                       String[]{ Manifest.permission. WRITE_EXTERNAL_STORAGE }, 1);
                } else {
                    openAlbum();
                }
            }
        });
    }

    private void openAlbum() {
        Intent intent = new Intent("android.intent.action.GET_CONTENT");
        intent.setType("image/*");
        startActivityForResult(intent, CHOOSE_PHOTO); // 打开相册
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
        int[] grantResults) {
        switch (requestCode) {
            case 1:
                if (grantResults.length > 0 && grantResults[0] == PackageManager.
                    PERMISSION_GRANTED) {
                    openAlbum();
                } else {
                    Toast.makeText(this, "You denied the permission",
                       Toast.LENGTH_SHORT).show();
                }
                break;
```

```
            default:
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch (requestCode) {
            ...
            case CHOOSE_PHOTO:
                if (resultCode == RESULT_OK) {
                    // 判断手机系统版本号
                    if (Build.VERSION.SDK_INT >= 19) {
                        // 4.4 及以上系统使用这个方法处理图片
                        handleImageOnKitKat(data);
                    } else {
                        // 4.4 以下系统使用这个方法处理图片
                        handleImageBeforeKitKat(data);
                    }
                }
                break;
            default:
                break;
        }
    }

    @TargetApi(19)
    private void handleImageOnKitKat(Intent data) {
        String imagePath = null;
        Uri uri = data.getData();
        if (DocumentsContract.isDocumentUri(this, uri)) {
            // 如果是 document 类型的 Uri，则通过 document id 处理
            String docId = DocumentsContract.getDocumentId(uri);
            if("com.android.providers.media.documents".equals(uri.getAuthority())) {
                String id = docId.split(":")[1]; // 解析出数字格式的 id
                String selection = MediaStore.Images.Media._ID + "=" + id;
                imagePath = getImagePath(MediaStore.Images.Media.EXTERNAL_
                   CONTENT_URI, selection);
            } else if ("com.android.providers.downloads.documents".equals(uri.
                getAuthority())) {
                Uri contentUri = ContentUris.withAppendedId(Uri.parse("content:
                   //downloads/public_downloads"), Long.valueOf(docId));
                imagePath = getImagePath(contentUri, null);
            }
        } else if ("content".equalsIgnoreCase(uri.getScheme())) {
            // 如果是 content 类型的 Uri，则使用普通方式处理
            imagePath = getImagePath(uri, null);
        } else if ("file".equalsIgnoreCase(uri.getScheme())) {
            // 如果是 file 类型的 Uri，直接获取图片路径即可
            imagePath = uri.getPath();
        }
        displayImage(imagePath); // 根据图片路径显示图片
    }

    private void handleImageBeforeKitKat(Intent data) {
        Uri uri = data.getData();
```

```
        String imagePath = getImagePath(uri, null);
        displayImage(imagePath);
    }

    private String getImagePath(Uri uri, String selection) {
        String path = null;
        // 通过Uri和selection来获取真实的图片路径
        Cursor cursor = getContentResolver().query(uri, null, selection, null, null);
        if (cursor != null) {
            if (cursor.moveToFirst()) {
                path = cursor.getString(cursor.getColumnIndex(MediaStore.
                    Images.Media.DATA));
            }
            cursor.close();
        }
        return path;
    }

    private void displayImage(String imagePath) {
        if (imagePath != null) {
            Bitmap bitmap = BitmapFactory.decodeFile(imagePath);
            picture.setImageBitmap(bitmap);
        } else {
            Toast.makeText(this, "failed to get image", Toast.LENGTH_SHORT).show();
        }
    }

}
```

可以看到，在 Choose From Album 按钮的点击事件里我们先是进行了一个运行时权限处理，动态申请 WRITE_EXTERNAL_STORAGE 这个危险权限。为什么需要申请这个权限呢？因为相册中的照片都是存储在 SD 卡上的，我们要从 SD 卡中读取照片就需要申请这个权限。WRITE_EXTERNAL_STORAGE 表示同时授予程序对 SD 卡读和写的能力。

当用户授权了权限申请之后会调用 openAlbum()方法，这里我们先是构建出了一个 Intent 对象，并将它的 action 指定为 android.intent.action.GET_CONTENT。接着给这个 Intent 对象设置一些必要的参数，然后调用 startActivityForResult()方法就可以打开相册程序选择照片了。注意在调用 startActivityForResult()方法的时候，我们给第二个参数传入的值变成了 CHOOSE_PHOTO，这样当从相册选择完图片回到 onActivityResult()方法时，就会进入 CHOOSE_PHOTO 的 case 来处理图片。接下来的逻辑就比较复杂了，首先为了兼容新老版本的手机，我们做了一个判断，如果是 4.4 及以上系统的手机就调用 handleImageOnKitKat()方法来处理图片，否则就调用 handleImageBeforeKitKat()方法来处理图片。之所以要这样做，是因为 Android 系统从 4.4 版本开始，选取相册中的图片不再返回图片真实的 Uri 了，而是一个封装过的 Uri，因此如果是 4.4 版本以上的手机就需要对这个 Uri 进行解析才行。

那么 handleImageOnKitKat()方法中的逻辑就基本是如何解析这个封装过的 Uri 了。这里有好几种判断情况，如果返回的 Uri 是 document 类型的话，那就取出 document id 进行处理，

如果不是的话，那就使用普通的方式处理。另外，如果 Uri 的 authority 是 media 格式的话，`document id` 还需要再进行一次解析，要通过字符串分割的方式取出后半部分才能得到真正的数字 id。取出的 id 用于构建新的 Uri 和条件语句，然后把这些值作为参数传入到 `getImagePath()`方法当中，就可以获取到图片的真实路径了。拿到图片的路径之后，再调用 `displayImage()`方法将图片显示到界面上。

相比于 `handleImageOnKitKat()`方法，`handleImageBeforeKitKat()`方法中的逻辑就要简单得多了，因为它的 Uri 是没有封装过的，不需要任何解析，直接将 Uri 传入到 `getImagePath()`方法当中就能获取到图片的真实路径了，最后同样是调用 `displayImage()`方法来让图片显示到界面上。

现在将程序重新运行到手机上，然后点击一下 Choose From Album 按钮，首先会弹出权限申请框，如图 8.14 所示。

点击允许之后就会打开手机相册，如图 8.15 所示。

然后随意选择一张照片，回到我们程序的界面，选中的照片应该就会显示出来了，如图 8.16 所示。

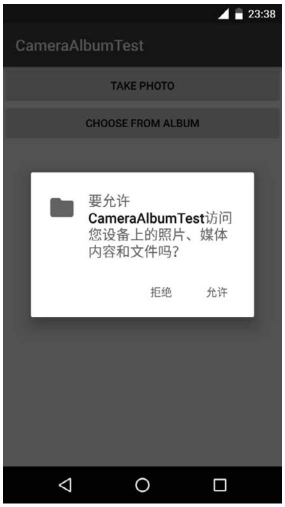


图 8.14　申请访问 SD 卡权限

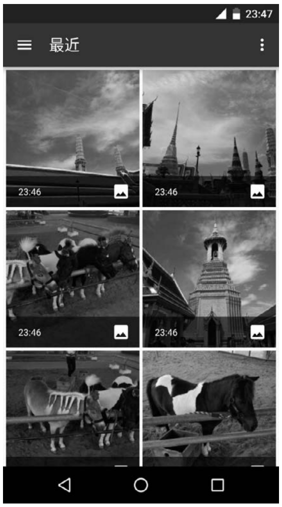


图 8.15　打开手机相册

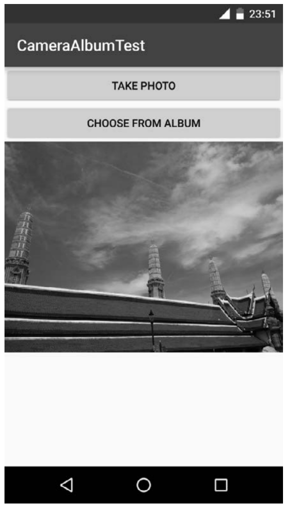


图 8.16　选择照片的最终效果

调用摄像头拍照以及从相册中选择照片是很多 Android 应用都会带有的功能，现在你已经将这两种技术都学会了，将来在工作中如果需要开发类似的功能，相信你一定能轻松完成的。不过目前我们的实现还不算完美，因为某些照片即使经过裁剪后体积仍然很大，直接加载到内存中有可能会导致程序崩溃。更好的做法是根据项目的需求先对照片进行适当的压缩，然后再加载到内存中。至于如何对照片进行压缩，就要考验你查阅资料的能力了，这里就不再展开进行讲解了。

8.4 播放多媒体文件

手机上最常见的休闲方式毫无疑问就是听音乐和看电影了，随着移动设备的普及，越来越多的人都可以随时享受优美的音乐，以及观看精彩的电影。而 Android 在播放音频和视频方面也是做了相当不错的支持，它提供了一套较为完整的 API，使得开发者可以很轻松地编写出一个简易的音频或视频播放器，下面我们就来具体地学习一下。

8.4.1 播放音频

在 Android 中播放音频文件一般都是使用 MediaPlayer 类来实现的，它对多种格式的音频文件提供了非常全面的控制方法，从而使得播放音乐的工作变得十分简单。下表列出了 MediaPlayer 类中一些较为常用的控制方法。

方法名	功能描述
setDataSource()	设置要播放的音频文件的位置
prepare()	在开始播放之前调用这个方法完成准备工作
start()	开始或继续播放音频
pause()	暂停播放音频
reset()	将MediaPlayer对象重置到刚刚创建的状态
seekTo()	从指定的位置开始播放音频
stop()	停止播放音频。调用这个方法后的MediaPlayer对象无法再播放音频
release()	释放掉与MediaPlayer对象相关的资源
isPlaying()	判断当前MediaPlayer是否正在播放音频
getDuration()	获取载入的音频文件的时长

简单了解了上述方法后，我们再来梳理一下 MediaPlayer 的工作流程。首先需要创建出一个 MediaPlayer 对象，然后调用 setDataSource()方法来设置音频文件的路径，再调用 prepare()方法使 MediaPlayer 进入到准备状态，接下来调用 start()方法就可以开始播放音频，调用 pause()方法就会暂停播放，调用 reset()方法就会停止播放。

下面就让我们通过一个具体的例子来学习一下吧，新建一个 PlayAudioTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/play"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Play" />
```

```
    <Button
        android:id="@+id/pause"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pause" />

    <Button
        android:id="@+id/stop"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Stop" />

</LinearLayout>
```

布局文件中放置了 3 个按钮，分别用于对音频文件进行播放、暂停和停止操作。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener{

    private MediaPlayer mediaPlayer = new MediaPlayer();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button play = (Button) findViewById(R.id.play);
        Button pause = (Button) findViewById(R.id.pause);
        Button stop = (Button) findViewById(R.id.stop);
        play.setOnClickListener(this);
        pause.setOnClickListener(this);
        stop.setOnClickListener(this);
        if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest. permission.
           WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(MainActivity.this, new String[]{
                Manifest.permission. WRITE_EXTERNAL_STORAGE }, 1);
        } else {
            initMediaPlayer(); // 初始化 MediaPlayer
        }
    }

    private void initMediaPlayer() {
        try {
            File file = new File(Environment.getExternalStorageDirectory(),
                "music.mp3");
            mediaPlayer.setDataSource(file.getPath()); // 指定音频文件的路径
            mediaPlayer.prepare(); // 让 MediaPlayer 进入到准备状态
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
        int[] grantResults) {
```

```
        switch (requestCode) {
            case 1:
                if (grantResults.length > 0 && grantResults[0] == PackageManager.
                    PERMISSION_GRANTED) {
                    initMediaPlayer();
                } else {
                    Toast.makeText(this, "拒绝权限将无法使用程序",
                    Toast.LENGTH_SHORT).show();
                    finish();
                }
                break;
            default:
        }
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.play:
                if (!mediaPlayer.isPlaying()) {
                    mediaPlayer.start(); // 开始播放
                }
                break;
            case R.id.pause:
                if (mediaPlayer.isPlaying()) {
                    mediaPlayer.pause(); // 暂停播放
                }
                break;
            case R.id.stop:
                if (mediaPlayer.isPlaying()) {
                    mediaPlayer.reset(); // 停止播放
                    initMediaPlayer();
                }
                break;
            default:
                break;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (mediaPlayer != null) {
            mediaPlayer.stop();
            mediaPlayer.release();
        }
    }

}
```

可以看到，在类初始化的时候我们就先创建了一个 MediaPlayer 的实例，然后在 `onCreate()` 方法中进行了运行时权限处理，动态申请 `WRITE_EXTERNAL_STORAGE` 权限。这是由于待会我们会在 SD 卡中放置一个音频文件，程序为了播放这个音频文件必须拥有访问 SD 卡的权限才行。

注意，在 onRequestPermissionsResult()方法中，如果用户拒绝了权限申请，那么就调用 finish()方法将程序直接关掉，因为如果没有 SD 卡的访问权限，我们这个程序将什么都干不了。

用户同意授权之后就会调用 initMediaPlayer()方法为 MediaPlayer 对象进行初始化操作。在 initMediaPlayer()方法中，首先是通过创建一个 File 对象来指定音频文件的路径，从这里可以看出，我们需要事先在 SD 卡的根目录下放置一个名为 music.mp3 的音频文件。后面依次调用了 setDataSource()方法和 prepare()方法，为 MediaPlayer 做好了播放前的准备。

接下来我们看一下各个按钮的点击事件中的代码。当点击 Play 按钮时会进行判断，如果当前 MediaPlayer 没有正在播放音频，则调用 start()方法开始播放。当点击 Pause 按钮时会判断，如果当前 MediaPlayer 正在播放音频，则调用 pause()方法暂停播放。当点击 Stop 按钮时会判断，如果当前 MediaPlayer 正在播放音频，则调用 reset()方法将 MediaPlayer 重置为刚刚创建的状态，然后重新调用一遍 initMediaPlayer()方法。

最后在 onDestroy()方法中，我们还需要分别调用 stop()方法和 release()方法，将与 MediaPlayer 相关的资源释放掉。

另外，千万不要忘记在 AndroidManifest.xml 文件中声明用到的权限，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.playaudiotest">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

这样一个简易版的音乐播放器就完成了，现在将程序运行到手机上会先弹出权限申请框，如图 8.17 所示。



图 8.17　音乐播放器主界面

同意授权之后就可以开始播放音乐了，点击一下 Play 按钮，优美的音乐就会响起，然后点击 Pause 按钮，音乐就会停住，再次点击 Play 按钮，会接着暂停之前的位置继续播放。这时如果点击一下 Stop 按钮，音乐也会停住，但是当再次点击 Play 按钮时，音乐就会从头开始播放了。

8.4.2　播放视频

播放视频文件其实并不比播放音频文件复杂，主要是使用 VideoView 类来实现的。这个类将视频的显示和控制集于一身，使得我们仅仅借助它就可以完成一个简易的视频播放器。VideoView 的用法和 MediaPlayer 也比较类似，主要有以下常用方法：

方　法　名	功能描述
setVideoPath()	设置要播放的视频文件的位置
start()	开始或继续播放视频
pause()	暂停播放视频
resume()	将视频重头开始播放
seekTo()	从指定的位置开始播放视频
isPlaying()	判断当前是否正在播放视频
getDuration()	获取载入的视频文件的时长

那么我们还是通过一个实际的例子来学习一下吧，新建 PlayVideoTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/play"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Play" />

        <Button
            android:id="@+id/pause"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Pause" />

        <Button
            android:id="@+id/replay"
            android:layout_width="0dp"
```

```
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Replay" />

    </LinearLayout>

    <VideoView
        android:id="@+id/video_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

在这个布局文件中，首先放置了 3 个按钮，分别用于控制视频的播放、暂停和重新播放。然后在按钮下面又放置了一个 VideoView，稍后的视频就将在这里显示。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener{

    private VideoView videoView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        videoView = (VideoView) findViewById(R.id.video_view);
        Button play = (Button) findViewById(R.id.play);
        Button pause = (Button) findViewById(R.id.pause);
        Button replay = (Button) findViewById(R.id.replay);
        play.setOnClickListener(this);
        pause.setOnClickListener(this);
        replay.setOnClickListener(this);
        if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.
            permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(MainActivity.this, new String[]{
                Manifest.permission. WRITE_EXTERNAL_STORAGE }, 1);
        } else {
            initVideoPath(); // 初始化 VideoView
        }
    }

    private void initVideoPath() {
        File file = new File(Environment.getExternalStorageDirectory(), "movie.mp4");
        videoView.setVideoPath(file.getPath()); // 指定视频文件的路径
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
        int[] grantResults) {
        switch (requestCode) {
            case 1:
                if (grantResults.length > 0 && grantResults[0] == PackageManager.
                    PERMISSION_GRANTED) {
```

```
                initVideoPath();
            } else {
                Toast.makeText(this, "拒绝权限将无法使用程序", Toast.LENGTH_SHORT).
                    show();
                finish();
            }
            break;
        default:
    }
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.play:
            if (!videoView.isPlaying()) {
                videoView.start(); // 开始播放
            }
            break;
        case R.id.pause:
            if (videoView.isPlaying()) {
                videoView.pause(); // 暂停播放
            }
            break;
        case R.id.replay:
            if (videoView.isPlaying()) {
                videoView.resume(); // 重新播放
            }
            break;
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    if (videoView != null) {
        videoView.suspend();
    }
}

}
```

这部分代码相信你理解起来会很轻松，因为它和前面播放音频的代码非常类似。首先在 `onCreate()`方法中同样进行了一个运行时权限处理，因为视频文件将会放在 SD 卡上。当用户同意授权了之后就会调用 `initVideoPath()`方法来设置视频文件的路径，这里我们需要事先在 SD 卡的根目录下放置一个名为 movie.mp4 的视频文件。

下面看一下各个按钮的点击事件中的代码。当点击 Play 按钮时会进行判断，如果当前并没有正在播放视频，则调用 `start()`方法开始播放。当点击 Pause 按钮时会判断，如果当前视频正在播放，则调用 `pause()`方法暂停播放。当点击 Replay 按钮时会判断，如果当前视频正在播放，则调用 `resume()`方法从头播放视频。

最后在 onDestroy()方法中，我们还需要调用一下 suspend()方法，将 VideoView 所占用的资源释放掉。

另外，仍然始终要记得在 AndroidManifest.xml 文件中声明用到的权限，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.playvideotest">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

现在将程序运行到手机上，会先弹出一个权限申请对话框，同意授权之后点击一下 Play 按钮，就可以看到视频已经开始播放了，如图 8.18 所示。

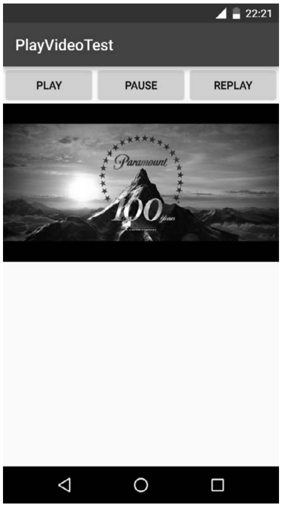


图 8.18　VideoView 播放视频的效果

点击 Pause 按钮可以暂停视频的播放，点击 Replay 按钮可以从头播放视频。

这样的话，你就已经将 VideoView 的基本用法掌握得差不多了。不过，为什么它的用法和 MediaPlayer 这么相似呢？其实 VideoView 只是帮我们做了一个很好的封装而已，它的背后仍然是使用 MediaPlayer 来对视频文件进行控制的。另外需要注意，VideoView 并不是一个万能的视频播放工具类，它在视频格式的支持以及播放效率方面都存在着较大的不足。所以，如果想要仅仅使用 VideoView 就编写出一个功能非常强大的视频播放器是不太现实的。但是如果只是用于播放一些游戏的片头动画，或者某个应用的视频宣传，使用 VideoView 还是绰绰有余的。

好了，关于 Android 多媒体方面的知识你已经学得足够多了，下面就让我们一起来总结一下本章所学的内容吧。

8.5 小结与点评

本章我们主要对 Android 系统中的各种多媒体技术进行了学习，其中包括通知的使用技巧、调用摄像头拍照、从相册中选取照片，以及播放音频和视频文件。由于所涉及的多媒体技术在模拟器上很难看得到效果，因此本章中还特意讲解了在 Android 手机上调试程序的方法。

又是充实饱满的一章啊！现在多媒体方面的知识已经学得足够多了，我希望你可以很好地将它们消化掉，尤其是与通知相关的内容，因为后面的学习当中还会用到它。目前我们所学的所有东西都仅仅是在本地上进行的，而实际上几乎市场上的每个应用都会涉及网络交互的部分，所以下一章中我们将会学习一下 Android 网络编程方面的内容。

第 9 章

看看精彩的世界——使用网络技术

如果你在玩手机的时候不能上网，那你一定会感到特别地枯燥乏味。没错，现在早已不是玩单机的时代了，无论是 PC、手机、平板，还是电视，几乎都会具备上网的功能，在可预见的未来，手表、眼镜、汽车等设备也会逐个加入到这个行列，21 世纪的确是互联网的时代。

当然，Android 手机肯定也是可以上网的，所以作为开发者，我们就需要考虑如何利用网络来编写出更加出色的应用程序，像 QQ、微博、微信等常见的应用都会大量使用网络技术。本章主要会讲述如何在手机端使用 HTTP 协议和服务器端进行网络交互，并对服务器返回的数据进行解析，这也是 Android 中最常使用到的网络技术，下面就让我们一起来学习一下吧。

9.1 WebView 的用法

有时候我们可能会碰到一些比较特殊的需求，比如说要求在应用程序里展示一些网页。相信每个人都知道，加载和显示网页通常都是浏览器的任务，但是需求里又明确指出，不允许打开系统浏览器，而我们当然也不可能自己去编写一个浏览器出来，这时应该怎么办呢？

不用担心，Android 早就已经考虑到了这种需求，并提供了一个 WebView 控件，借助它我们就可以在自己的应用程序里嵌入一个浏览器，从而非常轻松地展示各种各样的网页。

WebView 的用法也是相当简单，下面我们就通过一个例子来学习一下吧。新建一个 WebViewTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <WebView
        android:id="@+id/web_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

可以看到，我们在布局文件中使用到了一个新的控件：WebView。这个控件当然也就是用来显示网页的了，这里的写法很简单，给它设置了一个 id，并让它充满整个屏幕。

然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WebView webView = (WebView) findViewById(R.id.web_view);
        webView.getSettings().setJavaScriptEnabled(true);
        webView.setWebViewClient(new WebViewClient());
        webView.loadUrl("http://www.baidu.com");
    }

}
```

MainActivity 中的代码也很短，首先使用 `findViewById()`方法获取到了 WebView 的实例，然后调用 WebView 的 `getSettings()`方法可以去设置一些浏览器的属性，这里我们并不去设置过多的属性，只是调用了 `setJavaScriptEnabled()`方法来让 WebView 支持 JavaScript 脚本。

接下来是非常重要的一个部分，我们调用了 WebView 的 `setWebViewClient()`方法，并传入了一个 WebViewClient 的实例。这段代码的作用是，当需要从一个网页跳转到另一个网页时，我们希望目标网页仍然在当前 WebView 中显示，而不是打开系统浏览器。

最后一步就非常简单了，调用 WebView 的 `loadUrl()`方法，并将网址传入，即可展示相应网页的内容，这里就让我们看一看百度的首页长什么样吧。

另外还需要注意，由于本程序使用到了网络功能，而访问网络是需要声明权限的，因此我们还得修改 AndroidManifest.xml 文件，并加入权限声明，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.webviewtest">

    <uses-permission android:name="android.permission.INTERNET" />

    ...
</manifest>
```

在开始运行之前，首先需要保证你的手机或模拟器是联网的，如果你使用的是模拟器，只需保证电脑能正常上网即可。然后就可以运行一下程序了，效果如图 9.1 所示。



图 9.1　WebView 加载网页

可以看到，WebViewTest 这个程序现在已经具备了一个简易浏览器的功能，不仅成功将百度的首页展示了出来，还可以通过点击链接浏览更多的网页。

当然，WebView 还有很多更加高级的使用技巧，我们就不再继续进行探讨了，因为那不是本章的重点。这里先介绍了一下 WebView 的用法，只是希望你能对 HTTP 协议的使用有一个最基本的认识，接下来我们就要利用这个协议来做一些真正的网络开发工作了。

9.2　使用 HTTP 协议访问网络

如果说真的要去深入分析 HTTP 协议，可能需要花费整整一本书的篇幅。这里我当然不会这么干，因为毕竟你是跟着我学习 Android 开发的，而不是网站开发。对于 HTTP 协议，你只需要稍微了解一些就足够了，它的工作原理特别简单，就是客户端向服务器发出一条 HTTP 请求，服务器收到请求之后会返回一些数据给客户端，然后客户端再对这些数据进行解析和处理就可以了。是不是非常简单？一个浏览器的基本工作原理也就是如此了。比如说上一节中使用到的 WebView 控件，其实也就是我们向百度的服务器发起了一条 HTTP 请求，接着服务器分析出我们想要访问的是百度的首页，于是会把该网页的 HTML 代码进行返回，然后 WebView 再调用手机浏览器的内核对返回的 HTML 代码进行解析，最终将页面展示出来。

简单来说，WebView 已经在后台帮我们处理好了发送 HTTP 请求、接收服务响应、解析返回数据，以及最终的页面展示这几步工作，不过由于它封装得实在是太好了，反而使得我们不能那么直观地看出 HTTP 协议到底是如何工作的。因此，接下来就让我们通过手动发送 HTTP 请求的方式，来更加深入地理解一下这个过程。

9.2.1 使用 HttpURLConnection

在过去，Android 上发送 HTTP 请求一般有两种方式：HttpURLConnection 和 HttpClient。不过由于 HttpClient 存在 API 数量过多、扩展困难等缺点，Android 团队越来越不建议我们使用这种方式。终于在 Android 6.0 系统中，HttpClient 的功能被完全移除了，标志着此功能被正式弃用，因此本小节我们就学习一下现在官方建议使用的 HttpURLConnection 的用法。

首先需要获取到 HttpURLConnection 的实例，一般只需 new 出一个 URL 对象，并传入目标的网络地址，然后调用一下 `openConnection()`方法即可，如下所示：

```
URL url = new URL("http://www.baidu.com");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
```

在得到了 HttpURLConnection 的实例之后，我们可以设置一下 HTTP 请求所使用的方法。常用的方法主要有两个：`GET` 和 `POST`。`GET` 表示希望从服务器那里获取数据，而 `POST` 则表示希望提交数据给服务器。写法如下：

```
connection.setRequestMethod("GET");
```

接下来就可以进行一些自由的定制了，比如设置连接超时、读取超时的毫秒数，以及服务器希望得到的一些消息头等。这部分内容根据自己的实际情况进行编写，示例写法如下：

```
connection.setConnectTimeout(8000);
connection.setReadTimeout(8000);
```

之后再调用 `getInputStream()`方法就可以获取到服务器返回的输入流了，剩下的任务就是对输入流进行读取，如下所示：

```
InputStream in = connection.getInputStream();
```

最后可以调用 `disconnect()`方法将这个 HTTP 连接关闭掉，如下所示：

```
connection.disconnect();
```

下面就让我们通过一个具体的例子来真正体验一下 HttpURLConnection 的用法。新建一个 NetworkTest 项目，首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/send_request"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send Request" />

    <ScrollView
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <TextView
            android:id="@+id/response_text"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </ScrollView>

</LinearLayout>
```

注意这里我们使用了一个新的控件：ScrollView，它是用来做什么的呢？由于手机屏幕的空间一般都比较小，有些时候过多的内容一屏是显示不下的，借助 ScrollView 控件的话，我们就可以以滚动的形式查看屏幕外的那部分内容。另外，布局中还放置了一个 Button 和一个 TextView，Button 用于发送 HTTP 请求，TextView 用于将服务器返回的数据显示出来。

接着修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    TextView responseText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button sendRequest = (Button) findViewById(R.id.send_request);
        responseText = (TextView) findViewById(R.id.response_text);
        sendRequest.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.send_request) {
            sendRequestWithHttpURLConnection();
        }
    }

    private void sendRequestWithHttpURLConnection() {
        // 开启线程来发起网络请求
        new Thread(new Runnable() {
            @Override
            public void run() {
                HttpURLConnection connection = null;
                BufferedReader reader = null;
                try {
                    URL url = new URL("https://www.baidu.com");
                    connection = (HttpURLConnection) url.openConnection();
                    connection.setRequestMethod("GET");
                    connection.setConnectTimeout(8000);
                    connection.setReadTimeout(8000);
                    InputStream in = connection.getInputStream();
```

```
                    // 下面对获取到的输入流进行读取
                    reader = new BufferedReader(new InputStreamReader(in));
                    StringBuilder response = new StringBuilder();
                    String line;
                    while ((line = reader.readLine()) != null) {
                        response.append(line);
                    }
                    showResponse(response.toString());
                } catch (Exception e) {
                    e.printStackTrace();
                } finally {
                    if (reader != null) {
                        try {
                            reader.close();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                    if (connection != null) {
                        connection.disconnect();
                    }
                }
            }
        }).start();
    }

    private void showResponse(final String response) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // 在这里进行UI操作，将结果显示到界面上
                responseText.setText(response);
            }
        });
    }

}
```

可以看到，我们在 Send Request 按钮的点击事件里调用了 `sendRequestWithHttpURLConnection()`方法，在这个方法中先是开启了一个子线程，然后在子线程里使用 HttpURLConnection 发出一条 HTTP 请求，请求的目标地址就是百度的首页。接着利用 BufferedReader 对服务器返回的流进行读取，并将结果传入到了 `showResponse()`方法中。而在 `showResponse()`方法里则是调用了一个 `runOnUiThread()`方法，然后在这个方法的匿名类参数中进行操作，将返回的数据显示到界面上。那么这里为什么要用这个 `runOnUiThread()`方法呢？这是因为 Android 是不允许在子线程中进行 UI 操作的，我们需要通过这个方法将线程切换到主线程，然后再更新 UI 元素。关于这部分内容，我们将会在下一章中进行详细讲解，现在你只需要记得必须这么写就可以了。

完整的一套流程就是这样，不过在开始运行之前，仍然别忘了要声明一下网络权限。修改

AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.networktest">

    <uses-permission android:name="android.permission.INTERNET" />

    ...

</manifest>
```

好了，现在运行一下程序，并点击 Send Request 按钮，结果如图 9.2 所示。



图 9.2　服务器响应的数据

是不是看得头晕眼花？没错，服务器返回给我们的就是这种 HTML 代码，只是通常情况下浏览器都会将这些代码解析成漂亮的网页后再展示出来。

那么如果是想要提交数据给服务器应该怎么办呢？其实也不复杂，只需要将 HTTP 请求的方法改成 POST，并在获取输入流之前把要提交的数据写出即可。注意每条数据都要以键值对的形式存在，数据与数据之间用“&”符号隔开，比如说我们想要向服务器提交用户名和密码，就可以这样写：

```
connection.setRequestMethod("POST");
DataOutputStream out = new DataOutputStream(connection.getOutputStream());
out.writeBytes("username=admin&password=123456");
```

好了，相信你已经将 HttpURLConnection 的用法很好地掌握了。

9.2.2 使用 OkHttp

当然我们并不是只能使用 HttpURLConnection，完全没有任何其他选择，事实上在开源盛行的今天，有许多出色的网络通信库都可以替代原生的 HttpURLConnection，而其中 OkHttp 无疑是做得最出色的一个。

OkHttp 是由鼎鼎大名的 Square 公司开发的,这个公司在开源事业上面贡献良多,除了 OkHttp 之外，还开发了像 Picasso、Retrofit 等著名的开源项目。OkHttp 不仅在接口封装上面做得简单易用，就连在底层实现上也是自成一派，比起原生的 HttpURLConnection，可以说是有过之而无不及，现在已经成了广大 Android 开发者首选的网络通信库。那么本小节我们就来学习一下 OkHttp 的用法，OkHttp 的项目主页地址是：https://github.com/square/okhttp。

在使用 OkHttp 之前，我们需要先在项目中添加 OkHttp 库的依赖。编辑 app/build.gradle 文件，在 dependencies 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'com.squareup.okhttp3:okhttp:3.4.1'
}
```

添加上述依赖会自动下载两个库，一个是 OkHttp 库，一个是 Okio 库，后者是前者的通信基础。其中 3.4.1 是我写本书时 OkHttp 的最新版本，你可以访问 OkHttp 的项目主页来查看当前最新的版本是多少。

下面我们来看一下 OkHttp 的具体用法，首先需要创建一个 OkHttpClient 的实例，如下所示：

```
OkHttpClient client = new OkHttpClient();
```

接下来如果想要发起一条 HTTP 请求，就需要创建一个 `Request` 对象：

```
Request request = new Request.Builder().build();
```

当然，上述代码只是创建了一个空的 `Request` 对象，并没有什么实际作用，我们可以在最终的 `build()`方法之前连缀很多其他方法来丰富这个 `Request` 对象。比如可以通过 `url()`方法来设置目标的网络地址，如下所示：

```
Request request = new Request.Builder()
        .url("http://www.baidu.com")
        .build();
```

之后调用 OkHttpClient 的 `newCall()`方法来创建一个 `Call` 对象，并调用它的 `execute()`方法来发送请求并获取服务器返回的数据，写法如下：

```
Response response = client.newCall(request).execute();
```

其中 Response 对象就是服务器返回的数据了，我们可以使用如下写法来得到返回的具体内容：

```
String responseData = response.body().string();
```

如果是发起一条 POST 请求会比 GET 请求稍微复杂一点，我们需要先构建出一个 Request Body 对象来存放待提交的参数，如下所示：

```
RequestBody requestBody = new FormBody.Builder()
        .add("username", "admin")
        .add("password", "123456")
        .build();
```

然后在 Request.Builder 中调用一下 post()方法，并将 RequestBody 对象传入：

```
Request request = new Request.Builder()
        .url("http://www.baidu.com")
        .post(requestBody)
        .build();
```

接下来的操作就和 GET 请求一样了，调用 execute()方法来发送请求并获取服务器返回的数据即可。

好了，OkHttp 的基本用法就先学到这里，本书中后面所有网络相关的功能我们都将会使用 OkHttp 来实现，到时候再进行进一步的学习。那么现在我们先把 NetworkTest 这个项目改用 OkHttp 的方式再实现一遍吧。

由于布局部分完全不用改动，所以现在直接修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.send_request) {
            sendRequestWithOkHttp();
        }
    }

    private void sendRequestWithOkHttp() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    OkHttpClient client = new OkHttpClient();
                    Request request = new Request.Builder()
                            .url("http://www.baidu.com")
                            .build();
                    Response response = client.newCall(request).execute();
                    String responseData = response.body().string();
                    showResponse(responseData);
                } catch (Exception e) {
                    e.printStackTrace();
```

```
                }
            }
        }).start();
    }

    ...
}
```

这里我们并没有做太多的改动，只是添加了一个 sendRequestWithOkHttp()方法，并在 Send Request 按钮的点击事件里去调用这个方法。在这个方法中同样还是先开启了一个子线程，然后在子线程里使用 OkHttp 发出一条 HTTP 请求，请求的目标地址还是百度的首页，OkHttp 的用法也正如前面所介绍的一样。最后仍然还是调用了 showResponse()方法来将服务器返回的数据显示到界面上。

仅仅是改了这么多代码，现在我们就可以重新运行一下程序了。点击 Send Request 按钮后，你会看到和上一小节中同样的运行结果，由此证明，使用 OkHttp 来发送 HTTP 请求的功能也已经成功实现了。

这样的话，相信你就已经把 HttpURLConnection 和 OkHttp 的基本用法都掌握得差不多了。

9.3　解析 XML 格式数据

通常情况下，每个需要访问网络的应用程序都会有一个自己的服务器，我们可以向服务器提交数据，也可以从服务器上获取数据。不过这个时候就出现了一个问题，这些数据到底要以什么样的格式在网络上传输呢？随便传递一段文本肯定是不行的，因为另一方根本就不会知道这段文本的用途是什么。因此，一般我们都会在网络上传输一些格式化后的数据，这种数据会有一定的结构规格和语义，当另一方收到数据消息之后就可以按照相同的结构规格进行解析，从而取出他想要的那部分内容。

在网络上传输数据时最常用的格式有两种：XML 和 JSON，下面我们就来一个一个地进行学习，本节首先学习一下如何解析 XML 格式的数据。

在开始之前我们还需要先解决一个问题，就是从哪儿才能获取一段 XML 格式的数据呢？这里我准备教你搭建一个最简单的 Web 服务器，在这个服务器上提供一段 XML 文本，然后我们在程序里去访问这个服务器，再对得到的 XML 文本进行解析。

搭建 Web 服务器其实非常简单，有很多的服务器类型可供选择，这里我准备使用 Apache 服务器。首先你需要去下载一个 Apache 服务器的安装包，官方下载地址是：http://httpd.apache.org/download.cgi。如果你在这个网址中找不到 Windows 版的安装包，也可以直接在百度上搜索“Apache 服务器下载”，将会找到很多下载链接。

下载完成后双击就可以进行安装了，如图 9.3 所示。



图 9.3　Apache 服务器安装界面

然后一直点击 Next，会提示让你输入自己的域名，我们随便填一个域名就可以了，如图 9.4 所示。

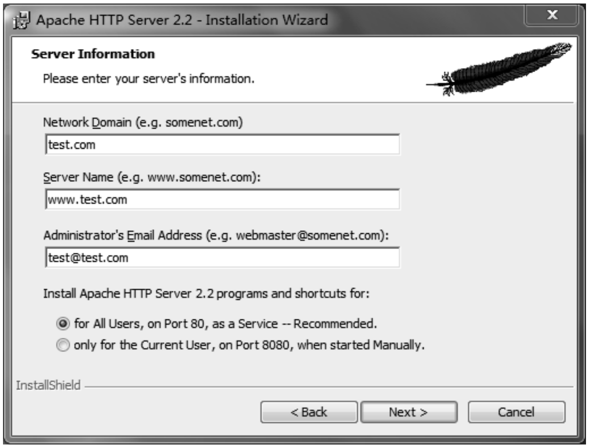


图 9.4　填入域名和服务器信息

接着继续一直点击 Next，会提示让你选择程序安装的路径，这里我选择安装到 C:\Apache 目录下，之后再继续点击 Next 就可以完成安装了。安装成功后服务器会自动启动起来，你可以打开电脑的浏览器来验证一下。在地址栏输入 127.0.0.1，如果出现了如图 9.5 所示的界面，就说明服务器已经启动成功了。



图 9.5 Apache 服务器的默认主页

接下来进入到 C:\Apache\htdocs 目录下，在这里新建一个名为 get_data.xml 的文件，然后编辑这个文件，并加入如下 XML 格式的内容。

```
<apps>
    <app>
        <id>1</id>
        <name>Google Maps</name>
        <version>1.0</version>
    </app>
    <app>
        <id>2</id>
        <name>Chrome</name>
        <version>2.1</version>
    </app>
    <app>
        <id>3</id>
        <name>Google Play</name>
        <version>2.3</version>
    </app>
</apps>
```

这时在浏览器中访问 http://127.0.0.1/get_data.xml 这个网址，就应该出现如图 9.6 所示的内容。

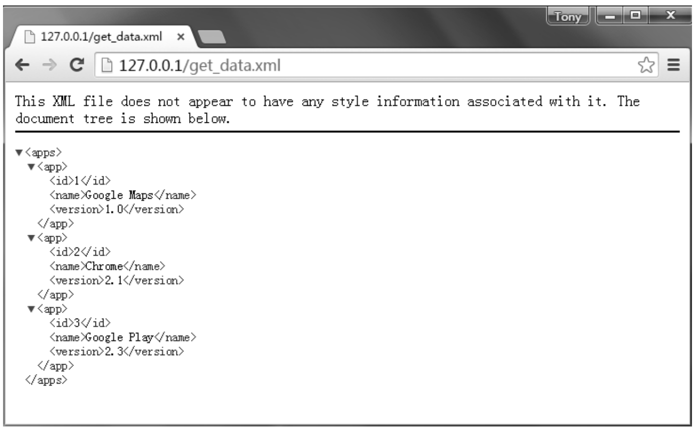


图 9.6 在浏览器验证 XML 数据

好了，准备工作到此结束，接下来就让我们在 Android 程序里去获取并解析这段 XML 数据吧。

9.3.1　Pull 解析方式

解析 XML 格式的数据其实也有挺多种方式的，本节中我们学习比较常用的两种，Pull 解析和 SAX 解析。那么简单起见，这里仍然是在 NetworkTest 项目的基础上继续开发，这样我们就可以重用之前网络通信部分的代码，从而把工作的重心放在 XML 数据解析上。

既然 XML 格式的数据已经提供好了，现在要做的就是从中解析出我们想要得到的那部分内容。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    private void sendRequestWithOkHttp() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    OkHttpClient client = new OkHttpClient();
                    Request request = new Request.Builder()
                             // 指定访问的服务器地址是电脑本机
                            .url("http://10.0.2.2/get_data.xml")
                            .build();
                    Response response = client.newCall(request).execute();
                    String responseData = response.body().string();
                    parseXMLWithPull(responseData);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }

    ...

    private void parseXMLWithPull(String xmlData) {
        try {
            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
            XmlPullParser xmlPullParser = factory.newPullParser();
            xmlPullParser.setInput(new StringReader(xmlData));
            int eventType = xmlPullParser.getEventType();
            String id = "";
            String name = "";
            String version = "";
            while (eventType != XmlPullParser.END_DOCUMENT) {
                String nodeName = xmlPullParser.getName();
                switch (eventType) {
                    // 开始解析某个节点
```

```
                    case XmlPullParser.START_TAG: {
                        if ("id".equals(nodeName)) {
                            id = xmlPullParser.nextText();
                        } else if ("name".equals(nodeName)) {
                            name = xmlPullParser.nextText();
                        } else if ("version".equals(nodeName)) {
                            version = xmlPullParser.nextText();
                        }
                        break;
                    }
                    // 完成解析某个节点
                    case XmlPullParser.END_TAG: {
                        if ("app".equals(nodeName)) {
                            Log.d("MainActivity", "id is " + id);
                            Log.d("MainActivity", "name is " + name);
                            Log.d("MainActivity", "version is " + version);
                        }
                        break;
                    }
                    default:
                        break;
                }
                eventType = xmlPullParser.next();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

可以看到，这里首先是将 HTTP 请求的地址改成了 http://10.0.2.2/get_data.xml，10.0.2.2 对于模拟器来说就是电脑本机的 IP 地址。在得到了服务器返回的数据后，我们并不再直接将其展示，而是调用了 `parseXMLWithPull()`方法来解析服务器返回的数据。

下面就来仔细看下 `parseXMLWithPull()`方法中的代码吧。这里首先要获取到一个 `XmlPullParserFactory` 的实例，并借助这个实例得到 `XmlPullParser` 对象，然后调用 `XmlPullParser` 的 `setInput()`方法将服务器返回的 XML 数据设置进去就可以开始解析了。解析的过程也非常简单，通过 `getEventType()`可以得到当前的解析事件，然后在一个 while 循环中不断地进行解析，如果当前的解析事件不等于 XmlPullParser.END_DOCUMENT，说明解析工作还没完成，调用 `next()`方法后可以获取下一个解析事件。

在 `while` 循环中，我们通过 `getName()`方法得到当前节点的名字，如果发现节点名等于 id、name 或 version，就调用 `nextText()`方法来获取节点内具体的内容，每当解析完一个 app 节点后就将获取到的内容打印出来。

好了，整体的过程就是这么简单，下面就让我们来测试一下吧。运行 NetworkTest 项目，然后点击 Send Request 按钮，观察 logcat 中的打印日志，如图 9.7 所示。

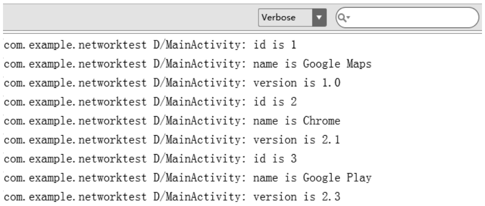


图 9.7　打印从 XML 中解析出的数据

可以看到，我们已经将 XML 数据中的指定内容成功解析出来了。

9.3.2　SAX 解析方式

Pull 解析方式虽然非常好用，但它并不是我们唯一的选择。SAX 解析也是一种特别常用的 XML 解析方式，虽然它的用法比 Pull 解析要复杂一些，但在语义方面会更加清楚。

通常情况下我们都会新建一个类继承自 DefaultHandler，并重写父类的 5 个方法，如下所示：

```
public class MyHandler extends DefaultHandler {

    @Override
    public void startDocument() throws SAXException {
    }

    @Override
    public void startElement(String uri, String localName, String qName, Attributes
        attributes) throws SAXException {
    }

    @Override
    public void characters(char[] ch, int start, int length) throws SAXException {
    }

    @Override
    public void endElement(String uri, String localName, String qName) throws
        SAXException {
    }

    @Override
    public void endDocument() throws SAXException {
    }

}
```

这 5 个方法一看就很清楚吧？startDocument()方法会在开始 XML 解析的时候调用，startElement()方法会在开始解析某个节点的时候调用，characters()方法会在获取节点中内容的时候调用，endElement()方法会在完成解析某个节点的时候调用，endDocument()方法会在完成整个 XML 解析的时候调用。其中，startElement()、characters()和 endElement()

这 3 个方法是有参数的，从 XML 中解析出的数据就会以参数的形式传入到这些方法中。需要注意的是，在获取节点中的内容时，`characters()`方法可能会被调用多次，一些换行符也被当作内容解析出来，我们需要针对这种情况在代码中做好控制。

那么下面就让我们尝试用 SAX 解析的方式来实现和上一小节中同样的功能吧。新建一个`ContentHandler`类继承自`DefaultHandler`，并重写父类的 5 个方法，如下所示：

```
public class ContentHandler extends DefaultHandler {

    private String nodeName;

    private StringBuilder id;

    private StringBuilder name;

    private StringBuilder version;

    @Override
    public void startDocument() throws SAXException {
        id = new StringBuilder();
        name = new StringBuilder();
        version = new StringBuilder();
    }

    @Override
    public void startElement(String uri, String localName, String qName, Attributes
        attributes) throws SAXException {
        // 记录当前节点名
        nodeName = localName;
    }

    @Override
    public void characters(char[] ch, int start, int length) throws SAXException {
        // 根据当前的节点名判断将内容添加到哪一个StringBuilder对象中
        if ("id".equals(nodeName)) {
            id.append(ch, start, length);
        } else if ("name".equals(nodeName)) {
            name.append(ch, start, length);
        } else if ("version".equals(nodeName)) {
            version.append(ch, start, length);
        }
    }

    @Override
    public void endElement(String uri, String localName, String qName) throws
        SAXException {
        if ("app".equals(localName)) {
            Log.d("ContentHandler", "id is " + id.toString().trim());
            Log.d("ContentHandler", "name is " + name.toString().trim());
            Log.d("ContentHandler", "version is " + version.toString().trim());
            // 最后要将StringBuilder清空掉
            id.setLength(0);
```

```
            name.setLength(0);
            version.setLength(0);
        }
    }

    @Override
    public void endDocument() throws SAXException {
        super.endDocument();
    }

}
```

可以看到，我们首先给 `id`、`name` 和 `version` 节点分别定义了一个 `StringBuilder` 对象，并在 `startDocument()`方法里对它们进行了初始化。每当开始解析某个节点的时候，`startElement()`方法就会得到调用，其中 `localName` 参数记录着当前节点的名字，这里我们把它记录下来。接着在解析节点中具体内容的时候就会调用 `characters()`方法，我们会根据当前的节点名进行判断，将解析出的内容添加到哪一个 `StringBuilder` 对象中。最后在 `endElement()`方法中进行判断，如果 app 节点已经解析完成，就打印出 `id`、`name` 和 `version` 的内容。需要注意的是，目前 `id`、`name` 和 `version` 中都可能是包括回车或换行符的，因此在打印之前我们还需要调用一下 `trim()`方法，并且打印完成后还要将 `StringBuilder` 的内容清空掉，不然的话会影响下一次内容的读取。

接下来的工作就非常简单了，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    private void sendRequestWithOkHttp() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    OkHttpClient client = new OkHttpClient();
                    Request request = new Request.Builder()
                            // 指定访问的服务器地址是电脑本机
                            .url("http://10.0.2.2/get_data.xml")
                            .build();
                    Response response = client.newCall(request).execute();
                    String responseData = response.body().string();
                    parseXMLWithSAX(responseData);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }

    ...
```

```
        private void parseXMLWithSAX(String xmlData) {
            try {
                SAXParserFactory factory = SAXParserFactory.newInstance();
                XMLReader xmlReader = factory.newSAXParser().getXMLReader();
                ContentHandler handler = new ContentHandler();
                // 将 ContentHandler 的实例设置到 XMLReader 中
                xmlReader.setContentHandler(handler);
                // 开始执行解析
                xmlReader.parse(new InputSource(new StringReader(xmlData)));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }
```

在得到了服务器返回的数据后，我们这次去调用 `parseXMLWithSAX()`方法来解析 XML 数据。`parseXMLWithSAX()`方法中先是创建了一个 `SAXParserFactory` 的对象，然后再获取到 `XMLReader` 对象，接着将我们编写的 ContentHandler 的实例设置到 XMLReader 中，最后调用 `parse()`方法开始执行解析就好了。

现在重新运行一下程序，点击 Send Request 按钮后观察 logcat 中的打印日志，你会看到和图 9.7 中一样的结果。

除了 Pull 解析和 SAX 解析之外，其实还有一种 DOM 解析方式也算挺常用的，不过这里我们就不再展开进行讲解了，感兴趣的话你可以自己去查阅一下相关资料。

9.4 解析 JSON 格式数据

现在你已经掌握了 XML 格式数据的解析方式,那么接下来我们要去学习一下如何解析 JSON 格式的数据了。比起 XML，JSON 的主要优势在于它的体积更小，在网络上传输的时候可以更省流量。但缺点在于，它的语义性较差，看起来不如 XML 直观。

在开始之前，我们还需要在 C:\Apache\htdocs 目录中新建一个 get_data.json 的文件，然后编辑这个文件，并加入如下 JSON 格式的内容：

```
[{"id":"5","version":"5.5","name":"Clash of Clans"},
{"id":"6","version":"7.0","name":"Boom Beach"},
{"id":"7","version":"3.5","name":"Clash Royale"}]
```

这时在浏览器中访问 http://127.0.0.1/get_data.json 这个网址,就应该出现如图 9.8 所示的内容。

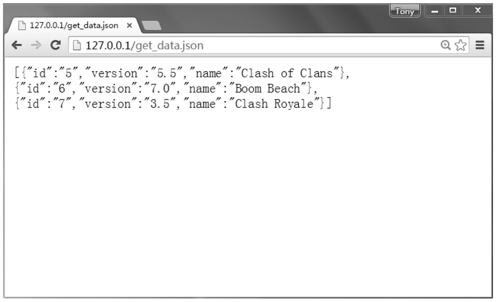


图 9.8　在浏览器验证 JSON 数据

好了，这样我们把 JSON 格式的数据也准备好了，下面就开始学习如何在 Android 程序中解析这些数据吧。

9.4.1　使用 JSONObject

类似地，解析 JSON 数据也有很多种方法，可以使用官方提供的 JSONObject，也可以使用谷歌的开源库 GSON。另外，一些第三方的开源库如 Jackson、FastJSON 等也非常不错。本节中我们就来学习一下前两种解析方式的用法。

修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    private void sendRequestWithOkHttp() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    OkHttpClient client = new OkHttpClient();
                    Request request = new Request.Builder()
                            // 指定访问的服务器地址是电脑本机
                            .url("http://10.0.2.2/get_data.json")
                            .build();
                    Response response = client.newCall(request).execute();
                    String responseData = response.body().string();
                    parseJSONWithJSONObject(responseData);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }

    ...
```

```
    private void parseJSONWithJSONObject(String jsonData) {
        try {
            JSONArray jsonArray = new JSONArray(jsonData);
            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject jsonObject = jsonArray.getJSONObject(i);
                String id = jsonObject.getString("id");
                String name = jsonObject.getString("name");
                String version = jsonObject.getString("version");
                Log.d("MainActivity", "id is " + id);
                Log.d("MainActivity", "name is " + name);
                Log.d("MainActivity", "version is " + version);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

首先记得要将 HTTP 请求的地址改成 http://10.0.2.2/get_data.json，然后在得到了服务器返回的数据后调用 `parseJSONWithJSONObject()`方法来解析数据。可以看到，解析 JSON 的代码真的非常简单，由于我们在服务器中定义的是一个 JSON 数组，因此这里首先是将服务器返回的数据传入到了一个 `JSONArray` 对象中。然后循环遍历这个 `JSONArray`，从中取出的每一个元素都是一个 `JSONObject` 对象，每个 `JSONObject` 对象中又会包含 `id`、`name` 和 `version` 这些数据。接下来只需要调用 `getString()`方法将这些数据取出，并打印出来即可。

好了，就是这么简单！现在重新运行一下程序，并点击 Send Request 按钮，结果如图 9.9 所示。

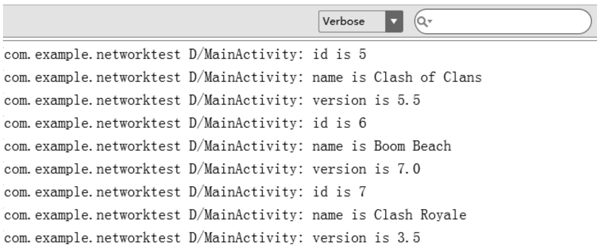


图 9.9　打印从 JSON 中解析出的数据

9.4.2　使用 GSON

如果你认为使用 JSONObject 来解析 JSON 数据已经非常简单了，那你就太容易满足了。谷歌提供的 GSON 开源库可以让解析 JSON 数据的工作简单到让你不敢想象的地步，那我们肯定是不能错过这个学习机会的。

不过 GSON 并没有被添加到 Android 官方的 API 中，因此如果想要使用这个功能的话，就必

须要在项目中添加 GSON 库的依赖。编辑 app/build.gradle 文件，在 dependencies 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.squareup.okhttp3:okhttp:3.4.1'
    compile 'com.google.code.gson:gson:2.7'
}
```

那么 GSON 库究竟是神奇在哪里呢？其实它主要就是可以将一段 JSON 格式的字符串自动映射成一个对象，从而不需要我们再手动去编写代码进行解析了。

比如说一段 JSON 格式的数据如下所示：

```
{"name":"Tom","age":20}
```

那我们就可以定义一个 `Person` 类，并加入 `name` 和 `age` 这两个字段，然后只需简单地调用如下代码就可以将 JSON 数据自动解析成一个 `Person` 对象了：

```
Gson gson = new Gson();
Person person = gson.fromJson(jsonData, Person.class);
```

如果需要解析的是一段 JSON 数组会稍微麻烦一点，我们需要借助 TypeToken 将期望解析成的数据类型传入到 `fromJson()`方法中，如下所示：

```
List<Person> people = gson.fromJson(jsonData, new TypeToken<List<Person>>()
{}.getType());
```

好了，基本的用法就是这样，下面就让我们来真正地尝试一下吧。首先新增一个 `App` 类，并加入 `id`、`name` 和 `version` 这 3 个字段，如下所示：

```
public class App {

    private String id;

    private String name;

    private String version;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
    }

}
```

然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    private void sendRequestWithOkHttp() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    OkHttpClient client = new OkHttpClient();
                    Request request = new Request.Builder()
                            // 指定访问的服务器地址是电脑本机
                            .url("http://10.0.2.2/get_data.json")
                            .build();
                    Response response = client.newCall(request).execute();
                    String responseData = response.body().string();
                    parseJSONWithGSON(responseData);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }

    ...

    private void parseJSONWithGSON(String jsonData) {
        Gson gson = new Gson();
        List<App> appList = gson.fromJson(jsonData, new TypeToken<List<App>>()
            {}.getType());
        for (App app : appList) {
            Log.d("MainActivity", "id is " + app.getId());
            Log.d("MainActivity", "name is " + app.getName());
            Log.d("MainActivity", "version is " + app.getVersion());
        }
    }

}
```

现在重新运行程序，点击 Send Request 按钮后观察 logcat 中的打印日志，你会看到和图 9.9 中一样的结果。

好了，这样我们就算是把 XML 和 JSON 这两种数据格式最常用的几种解析方法都学习完了，在网络数据的解析方面，你已经成功毕业了。

9.5　网络编程的最佳实践

目前你已经掌握了 HttpURLConnection 和 OkHttp 的用法，知道了如何发起 HTTP 请求，以及解析服务器返回的数据，但也许你还没有发现，之前我们的写法其实是很有问题的。因为一个应用程序很可能会在许多地方都使用到网络功能，而发送 HTTP 请求的代码基本都是相同的，如果我们每次都去编写一遍发送 HTTP 请求的代码，这显然是非常差劲的做法。

没错，通常情况下我们都应该将这些通用的网络操作提取到一个公共的类里，并提供一个静态方法，当想要发起网络请求的时候，只需简单地调用一下这个方法即可。比如使用如下的写法：

```
public class HttpUtil {

    public static String sendHttpRequest(String address) {
        HttpURLConnection connection = null;
        try {
            URL url = new URL(address);
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            connection.setConnectTimeout(8000);
            connection.setReadTimeout(8000);
            connection.setDoInput(true);
            connection.setDoOutput(true);
            InputStream in = connection.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));
            StringBuilder response = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                response.append(line);
            }
            return response.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return e.getMessage();
        } finally {
            if (connection != null) {
                connection.disconnect();
            }
        }
    }

}
```

以后每当需要发起一条 HTTP 请求的时候就可以这样写：

```
String address = "http://www.baidu.com";
String response = HttpUtil.sendHttpRequest(address);
```

在获取到服务器响应的数据后，我们就可以对它进行解析和处理了。但是需要注意，网络请求通常都是属于耗时操作，而 sendHttpRequest()方法的内部并没有开启线程，这样就有可能导致在调用 sendHttpRequest()方法的时候使得主线程被阻塞住。

你可能会说，很简单嘛，在 sendHttpRequest()方法内部开启一个线程不就解决这个问题了吗？其实没有你想象中的那么容易，因为如果我们在 sendHttpRequest()方法中开启了一个线程来发起 HTTP 请求，那么服务器响应的数据是无法进行返回的，所有的耗时逻辑都是在子线程里进行的，sendHttpRequest()方法会在服务器还没来得及响应的时候就执行结束了，当然也就无法返回响应的数据了。

那么遇到这种情况时应该怎么办呢？其实解决方法并不难，只需要使用 Java 的回调机制就可以了，下面就让我们来学习一下回调机制到底是如何使用的。

首先需要定义一个接口，比如将它命名成 HttpCallbackListener，代码如下所示：

```
public interface HttpCallbackListener {

    void onFinish(String response);

    void onError(Exception e);

}
```

可以看到，我们在接口中定义了两个方法，onFinish()方法表示当服务器成功响应我们请求的时候调用，onError()表示当进行网络操作出现错误的时候调用。这两个方法都带有参数，onFinish()方法中的参数代表着服务器返回的数据，而 onError()方法中的参数记录着错误的详细信息。

接着修改 HttpUtil 中的代码，如下所示：

```
public class HttpUtil {

    public static void sendHttpRequest(final String address, final
        HttpCallbackListener listener) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                HttpURLConnection connection = null;
                try {
                    URL url = new URL(address);
                    connection = (HttpURLConnection) url.openConnection();
                    connection.setRequestMethod("GET");
                    connection.setConnectTimeout(8000);
                    connection.setReadTimeout(8000);
                    connection.setDoInput(true);
                    connection.setDoOutput(true);
                    InputStream in = connection.getInputStream();
                    BufferedReader reader = new BufferedReader(new InputStreamReader
```

```
                            (in));
                        StringBuilder response = new StringBuilder();
                        String line;
                        while ((line = reader.readLine()) != null) {
                            response.append(line);
                        }
                        if (listener != null) {
                            // 回调onFinish()方法
                            listener.onFinish(response.toString());
                        }
                    } catch (Exception e) {
                        if (listener != null) {
                            // 回调onError()方法
                            listener.onError(e);
                        }
                    } finally {
                        if (connection != null) {
                            connection.disconnect();
                        }
                    }
                }
            }).start();
        }

}
```

我们首先给 `sendHttpRequest()`方法添加了一个 `HttpCallbackListener` 参数，并在方法的内部开启了一个子线程，然后在子线程里去执行具体的网络操作。注意，子线程中是无法通过 `return` 语句来返回数据的，因此这里我们将服务器响应的数据传入了 HttpCallbackListener 的 `onFinish()`方法中，如果出现了异常就将异常原因传入到 `onError()`方法中。

现在 `sendHttpRequest()`方法接收两个参数了，因此我们在调用它的时候还需要将 HttpCallbackListener 的实例传入，如下所示：

```
HttpUtil.sendHttpRequest(address, new HttpCallbackListener() {
    @Override
    public void onFinish(String response) {
        // 在这里根据返回内容执行具体的逻辑
    }

    @Override
    public void onError(Exception e) {
        // 在这里对异常情况进行处理
    }
});
```

这样的话，当服务器成功响应的时候，我们就可以在 `onFinish()`方法里对响应数据进行处理了。类似地，如果出现了异常，就可以在 `onError()`方法里对异常情况进行处理。如此一来，我们就巧妙地利用回调机制将响应数据成功返回给调用方了。

不过你会发现，上述使用 HttpURLConnection 的写法总体来说还是比较复杂的，那么使用

OkHttp 会变得简单吗？答案是肯定的，而且要简单得多，下面我们来具体看一下。在 HttpUtil 中加入一个 sendOkHttpRequest()方法，如下所示：

```
public class HttpUtil {

    ...

    public static void sendOkHttpRequest(String address, okhttp3.Callback callback) {
        OkHttpClient client = new OkHttpClient();
        Request request = new Request.Builder()
                .url(address)
                .build();
        client.newCall(request).enqueue(callback);
    }

}
```

可以看到，sendOkHttpRequest()方法中有一个 okhttp3.Callback 参数，这个是 OkHttp 库中自带的一个回调接口，类似于我们刚才自己编写的 HttpCallbackListener。然后在 client.newCall()之后没有像之前那样一直调用 execute()方法，而是调用了一个 enqueue()方法，并把 okhttp3.Callback 参数传入。相信聪明的你已经猜到了，OkHttp 在 enqueue()方法的内部已经帮我们开好子线程了，然后会在子线程中去执行 HTTP 请求，并将最终的请求结果回调到 okhttp3.Callback 当中。

那么我们在调用 sendOkHttpRequest()方法的时候就可以这样写：

```
HttpUtil.sendOkHttpRequest("http://www.baidu.com", new okhttp3.Callback() {

@Override
    public void onResponse(Call call, Response response) throws IOException {
        // 得到服务器返回的具体内容
        String responseData = response.body().string();
    }

    @Override
    public void onFailure(Call call, IOException e) {
        // 在这里对异常情况进行处理
    }

});
```

由此可以看出，OkHttp 的接口设计得确实非常人性化，它将一些常用的功能进行了很好的封装，使得我们只需编写少量的代码就能完成较为复杂的网络操作。当然这并不是 OkHttp 的全部，后面我们还会继续学习它的其他相关知识。

另外需要注意的是，不管是使用 HttpURLConnection 还是 OkHttp，最终的回调接口都还是在子线程中运行的，因此我们不可以在这里执行任何的 UI 操作，除非借助 runOnUiThread()方法来进行线程转换。至于具体的原因，我们很快就会在下一章中学习到了。

9.6　小结与点评

本章中我们主要学习了在 Android 中使用 HTTP 协议来进行网络交互的知识，虽然 Android 中支持的网络通信协议有很多种，但 HTTP 协议无疑是最常用的一种。通常我们有两种方式来发送 HTTP 请求，分别是 HttpURLConnection 和 OkHttp，相信这两种方式你都已经很好地掌握了。

接着我们又学习了 XML 和 JSON 格式数据的解析方式，因为服务器响应给我们的数据一般都是属于这两种格式的。无论是 XML 还是 JSON，它们各自又拥有多种解析方式，这里我们只是学习了最常用的几种，如果以后你的工作中还需要用到其他的解析方式，可以自行去学习。

本章的最后同样是最佳实践环节，在这次的最佳实践中，我们主要学习了如何利用 Java 的回调机制来将服务器响应的数据进行返回。其实除此之外，还有很多地方都可以使用到 Java 的回调机制，希望你能举一反三，以后在其他地方需要用到回调机制时都能够灵活地使用。

在进行了一章多媒体和一章网络的相关知识学习后，你是否想起来 Android 四大组件中还剩一个没有学过呢！那么下面就让我们进入到 Android 服务的学习旅程之中。

第 10 章

后台默默的劳动者——探究服务

记得在我上大学的时候，iPhone 是属于少数人才拥有的稀有物品，Android 甚至还没面世，那个时候全球的手机市场是由诺基亚统治着的。当时我觉得诺基亚的 Symbian 操作系统做得特别出色，因为比起一般的手机，它可以支持后台功能。那个时候能够一边打着电话、听着音乐，一边在后台挂着 QQ 是件非常酷的事情。所以我也曾经单纯地认为，支持后台的手机就是智能手机。

而如今，Symbian 早已风光不再，Android 和 iOS 几乎占据了智能手机全部的市场份额。在这两大移动操作系统中，iOS 一开始是不支持后台的，后来逐渐意识到这个功能的重要性，才加入了后台功能。而 Android 则是沿用了 Symbian 的老习惯，从一开始就支持后台功能，这使得应用程序即使在关闭的情况下仍然可以在后台继续运行。不管怎么说，后台功能属于四大组件之一，其重要程度不言而喻，那么我们自然要好好学习一下它的用法了。

10.1 服务是什么

服务（Service）是 Android 中实现程序后台运行的解决方案，它非常适合去执行那些不需要和用户交互而且还要求长期运行的任务。服务的运行不依赖于任何用户界面，即使程序被切换到后台，或者用户打开了另外一个应用程序，服务仍然能够保持正常运行。

不过需要注意的是，服务并不是运行在一个独立的进程当中的，而是依赖于创建服务时所在的应用程序进程。当某个应用程序进程被杀掉时，所有依赖于该进程的服务也会停止运行。

另外，也不要被服务的后台概念所迷惑，实际上服务并不会自动开启线程，所有的代码都是默认运行在主线程当中的。也就是说，我们需要在服务的内部手动创建子线程，并在这里执行具体的任务，否则就有可能出现主线程被阻塞住的情况。那么本章的第一堂课，我们就先来学习一下关于 Android 多线程编程的知识。

10.2　Android 多线程编程

熟悉 Java 的你，对多线程编程一定不会陌生吧。当我们需要执行一些耗时操作，比如说发起一条网络请求时，考虑到网速等其他原因，服务器未必会立刻响应我们的请求，如果不将这类操作放在子线程里去运行，就会导致主线程被阻塞住，从而影响用户对软件的正常使用。那么就让我们从线程的基本用法开始学习吧。

10.2.1　线程的基本用法

Android 多线程编程其实并不比 Java 多线程编程特殊，基本都是使用相同的语法。比如说，定义一个线程只需要新建一个类继承自 Thread，然后重写父类的 run()方法，并在里面编写耗时逻辑即可，如下所示：

```
class MyThread extends Thread {

    @Override
    public void run() {
        // 处理具体的逻辑
    }

}
```

那么该如何启动这个线程呢？其实也很简单，只需要 new 出 MyThread 的实例，然后调用它的 start()方法，这样 run()方法中的代码就会在子线程当中运行了，如下所示：

```
new MyThread().start();
```

当然，使用继承的方式耦合性有点高，更多的时候我们都会选择使用实现 Runnable 接口的方式来定义一个线程，如下所示：

```
class MyThread implements Runnable {

    @Override
    public void run() {
        // 处理具体的逻辑
    }

}
```

如果使用了这种写法，启动线程的方法也需要进行相应的改变，如下所示：

```
MyThread myThread = new MyThread();
new Thread(myThread).start();
```

可以看到，Thread 的构造函数接收一个 Runnable 参数，而我们 new 出的 MyThread 正是一个实现了 Runnable 接口的对象，所以可以直接将它传入到 Thread 的构造函数里。接着调用 Thread 的 start()方法，run()方法中的代码就会在子线程当中运行了。

当然，如果你不想专门再定义一个类去实现 Runnable 接口，也可以使用匿名类的方式，这种写法更为常见，如下所示：

```
new Thread(new Runnable() {

    @Override
    public void run() {
        // 处理具体的逻辑
    }

}).start();
```

以上几种线程的使用方式相信你都不会感到陌生，因为在 Java 中创建和启动线程也是使用同样的方式。了解了线程的基本用法后，下面我们来看一下 Android 多线程编程与 Java 多线程编程不同的地方。

10.2.2 在子线程中更新 UI

和许多其他的 GUI 库一样，Android 的 UI 也是线程不安全的。也就是说，如果想要更新应用程序里的 UI 元素，则必须在主线程中进行，否则就会出现异常。

眼见为实，让我们通过一个具体的例子来验证一下吧。新建一个 AndroidThreadTest 项目，然后修改 activity_main.xml 中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/change_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Change Text" />

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Hello world"
        android:textSize="20sp" />

</RelativeLayout>
```

布局文件中定义了两个控件，TextView 用于在屏幕的正中央显示一个 Hello world 字符串，Button 用于改变 TextView 中显示的内容，我们希望在点击 Button 后可以把 TextView 中显示的字符串改成 Nice to meet you。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private TextView text;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        text = (TextView) findViewById(R.id.text);
        Button changeText = (Button) findViewById(R.id.change_text);
        changeText.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.change_text:
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        text.setText("Nice to meet you");
                    }
                }).start();
                break;
            default:
                break;
        }
    }

}
```

可以看到，我们在 Change Text 按钮的点击事件里面开启了一个子线程，然后在子线程中调用 TextView 的 `setText()`方法将显示的字符串改成 Nice to meet you。代码的逻辑非常简单，只不过我们是在子线程中更新 UI 的。现在运行一下程序，并点击 Change Text 按钮，你会发现程序果然崩溃了，如图 10.1 所示。

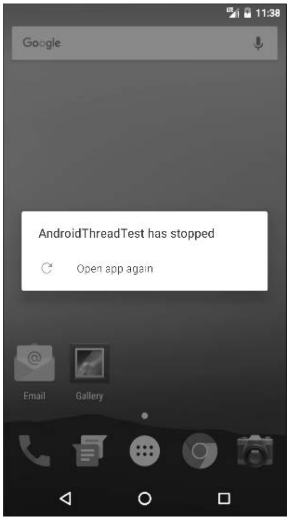


图 10.1 在子线程中更新 UI 导致崩溃

然后观察 logcat 中的错误日志，可以看出是由于在子线程中更新 UI 所导致的，如图 10.2 所示。

```
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original
thread that created a view hierarchy can touch its views.
```

图 10.2 崩溃的详细信息

由此证实了 Android 确实是不允许在子线程中进行 UI 操作的。但是有些时候，我们必须在子线程里去执行一些耗时任务，然后根据任务的执行结果来更新相应的 UI 控件，这该如何是好呢？

对于这种情况，Android 提供了一套异步消息处理机制，完美地解决了在子线程中进行 UI 操作的问题。本小节中我们先来学习一下异步消息处理的使用方法，下一小节中再去分析它的原理。

修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    public static final int UPDATE_TEXT = 1;

    private TextView text;

    private Handler handler = new Handler() {

        public void handleMessage(Message msg) {
            switch (msg.what) {
                case UPDATE_TEXT:
                    // 在这里可以进行 UI 操作
```

```
                    text.setText("Nice to meet you");
                    break;
                default:
                    break;
            }
        }

    };

    ...

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.change_text:
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        Message message = new Message();
                        message.what = UPDATE_TEXT;
                        handler.sendMessage(message); // 将Message对象发送出去
                    }
                }).start();
                break;
            default:
                break;
        }
    }

}
```

这里我们先是定义了一个整型常量 `UPDATE_TEXT`，用于表示更新 TextView 这个动作。然后新增一个 `Handler` 对象，并重写父类的 `handleMessage()`方法，在这里对具体的 Message 进行处理。如果发现 Message 的 `what` 字段的值等于 `UPDATE_TEXT`，就将 TextView 显示的内容改成 Nice to meet you。

下面再来看一下 Change Text 按钮的点击事件中的代码。可以看到，这次我们并没有在子线程里直接进行 UI 操作，而是创建了一个 `Message`（`android.os.Message`）对象，并将它的 `what` 字段的值指定为 `UPDATE_TEXT`，然后调用 `Handler` 的 `sendMessage()`方法将这条 Message 发送出去。很快，Handler 就会收到这条 Message，并在 `handleMessage()`方法中对它进行处理。注意此时 `handleMessage()`方法中的代码就是在主线程当中运行的了，所以我们可以放心地在这里进行 UI 操作。接下来对 Message 携带的 `what` 字段的值进行判断，如果等于 `UPDATE_TEXT`，就将 TextView 显示的内容改成 Nice to meet you。

现在重新运行程序，可以看到屏幕的正中央显示着 Hello world。然后点击一下 Change Text 按钮，显示的内容就被替换成 Nice to meet you，如图 10.3 所示。

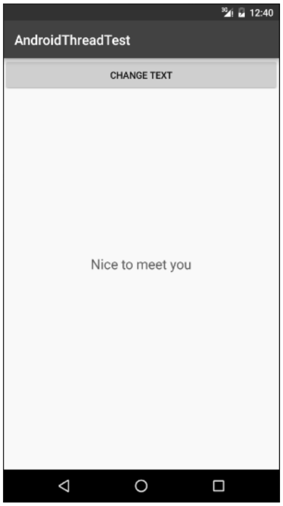


图 10.3　成功替换显示的文字

这样你就已经掌握了 Android 异步消息处理的基本用法，使用这种机制就可以出色地解决掉在子线程中更新 UI 的问题。不过恐怕你对它的工作原理还不是很清楚，下面我们就来分析一下 Android 异步消息处理机制到底是如何工作的。

10.2.3　解析异步消息处理机制

Android 中的异步消息处理主要由 4 个部分组成：Message、Handler、MessageQueue 和 Looper。其中 Message 和 Handler 在上一小节中我们已经接触过了，而 MessageQueue 和 Looper 对于你来说还是全新的概念，下面我就对这 4 个部分进行一下简要的介绍。

1. Message

Message 是在线程之间传递的消息，它可以在内部携带少量的信息，用于在不同线程之间交换数据。上一小节中我们使用到了 Message 的 `what` 字段，除此之外还可以使用 `arg1` 和 `arg2` 字段来携带一些整型数据，使用 `obj` 字段携带一个 `Object` 对象。

2. Handler

Handler 顾名思义也就是处理者的意思，它主要是用于发送和处理消息的。发送消息一般是使用 Handler 的 `sendMessage()`方法，而发出的消息经过一系列地辗转处理后，最终会传递到 Handler 的 `handleMessage()`方法中。

3. MessageQueue

MessageQueue 是消息队列的意思，它主要用于存放所有通过 Handler 发送的消息。这部分消息会一直存在于消息队列中，等待被处理。每个线程中只会有一个 `MessageQueue` 对象。

4. Looper

Looper 是每个线程中的 MessageQueue 的管家，调用 Looper 的 `loop()`方法后，就会进入到一个无限循环当中，然后每当发现 MessageQueue 中存在一条消息，就会将它取出，并传递到 Handler 的 `handleMessage()`方法中。每个线程中也只会有一个 `Looper` 对象。

了解了 Message、Handler、MessageQueue 以及 Looper 的基本概念后，我们再来把异步消息处理的整个流程梳理一遍。首先需要在主线程当中创建一个 `Handler` 对象，并重写 `handleMessage()`方法。然后当子线程中需要进行 UI 操作时，就创建一个 `Message` 对象，并通过 Handler 将这条消息发送出去。之后这条消息会被添加到 MessageQueue 的队列中等待被处理，而 Looper 则会一直尝试从 MessageQueue 中取出待处理消息，最后分发回 Handler 的 `handleMessage()`方法中。由于 Handler 是在主线程中创建的，所以此时 `handleMessage()`方法中的代码也会在主线程中运行，于是我们在这里就可以安心地进行 UI 操作了。整个异步消息处理机制的流程示意图如图 10.4 所示。

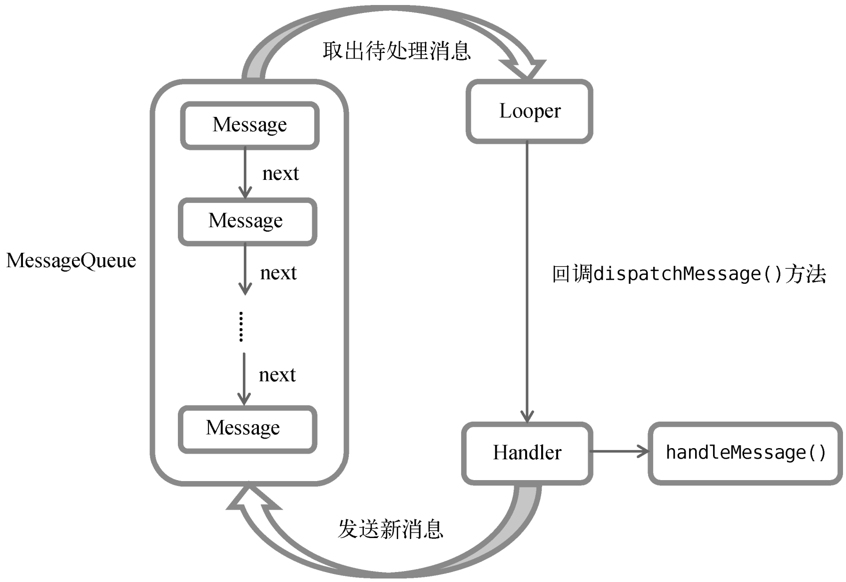


图 10.4 异步消息处理机制流程示意图

一条 Message 经过这样一个流程的辗转调用后，也就从子线程进入到了主线程，从不能更新 UI 变成了可以更新 UI，整个异步消息处理的核心思想也就是如此。

而我们在 9.2.1 小节中使用到的 `runOnUiThread()`方法其实就是一个异步消息处理机制的接口封装，它虽然表面上看起来用法更为简单，但其实背后的实现原理和图 10.4 中的描述是一模一样的。

10.2.4　使用 AsyncTask

不过为了更加方便我们在子线程中对 UI 进行操作，Android 还提供了另外一些好用的工具，比如 AsyncTask。借助 AsyncTask，即使你对异步消息处理机制完全不了解，也可以十分简单地从子线程切换到主线程。当然，AsyncTask 背后的实现原理也是基于异步消息处理机制的，只是 Android 帮我们做了很好的封装而已。

首先来看一下 AsyncTask 的基本用法，由于 AsyncTask 是一个抽象类，所以如果我们想使用它，就必须要创建一个子类去继承它。在继承时我们可以为 AsyncTask 类指定 3 个泛型参数，这 3 个参数的用途如下。

- Params。在执行 AsyncTask 时需要传入的参数，可用于在后台任务中使用。
- Progress。后台任务执行时，如果需要在界面上显示当前的进度，则使用这里指定的泛型作为进度单位。
- Result。当任务执行完毕后，如果需要对结果进行返回，则使用这里指定的泛型作为返回值类型。

因此，一个最简单的自定义 AsyncTask 就可以写成如下方式：

```
class DownloadTask extends AsyncTask<Void, Integer, Boolean> {
    ...
}
```

这里我们把 AsyncTask 的第一个泛型参数指定为 Void，表示在执行 AsyncTask 的时候不需要传入参数给后台任务。第二个泛型参数指定为 Integer，表示使用整型数据来作为进度显示单位。第三个泛型参数指定为 Boolean，则表示使用布尔型数据来反馈执行结果。

当然，目前我们自定义的 DownloadTask 还是一个空任务，并不能进行任何实际的操作，我们还需要去重写 AsyncTask 中的几个方法才能完成对任务的定制。经常需要去重写的方法有以下 4 个。

1. onPreExecute()

这个方法会在后台任务开始执行之前调用，用于进行一些界面上的初始化操作，比如显示一个进度条对话框等。

2. doInBackground(Params...)

这个方法中的所有代码都会在子线程中运行，我们应该在这里去处理所有的耗时任务。任务一旦完成就可以通过 return 语句来将任务的执行结果返回，如果 AsyncTask 的第三个泛型参数指定的是 Void，就可以不返回任务执行结果。注意，在这个方法中是不可以进行 UI 操作的，如果需要更新 UI 元素，比如说反馈当前任务的执行进度，可以调用 publishProgress (Progress...)方法来完成。

3. onProgressUpdate(Progress...)

当在后台任务中调用了 publishProgress(Progress...)方法后，onProgressUpdate (Progress...)方法就会很快被调用，该方法中携带的参数就是在后台任务中传递过来的。在这个方法中可以对 UI 进行操作，利用参数中的数值就可以对界面元素进行相应的更新。

4. onPostExecute(Result)

当后台任务执行完毕并通过 return 语句进行返回时，这个方法就很快会被调用。返回的数据会作为参数传递到此方法中，可以利用返回的数据来进行一些 UI 操作，比如说提醒任务执行的结果，以及关闭掉进度条对话框等。

因此，一个比较完整的自定义 AsyncTask 就可以写成如下方式：

```
class DownloadTask extends AsyncTask<Void, Integer, Boolean> {

    @Override
    protected void onPreExecute() {
        progressDialog.show(); // 显示进度对话框
    }

    @Override
    protected Boolean doInBackground(Void... params) {
        try {
            while (true) {
                int downloadPercent = doDownload(); // 这是一个虚构的方法
                publishProgress(downloadPercent);
                if (downloadPercent >= 100) {
                    break;
                }
            }
        } catch (Exception e) {
            return false;
        }
        return true;
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        // 在这里更新下载进度
        progressDialog.setMessage("Downloaded " + values[0] + "%");
    }

    @Override
    protected void onPostExecute(Boolean result) {
        progressDialog.dismiss(); // 关闭进度对话框
        // 在这里提示下载结果
        if (result) {
            Toast.makeText(context, "Download succeeded", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(context, " Download failed", Toast.LENGTH_SHORT).show();
        }
```

```
    }
}
```

在这个 DownloadTask 中，我们在 `doInBackground()`方法里去执行具体的下载任务。这个方法里的代码都是在子线程中运行的，因而不会影响到主线程的运行。注意这里虚构了一个 `doDownload()`方法，这个方法用于计算当前的下载进度并返回，我们假设这个方法已经存在了。在得到了当前的下载进度后，下面就该考虑如何把它显示到界面上了，由于 `doInBackground()`方法是在子线程中运行的，在这里肯定不能进行 UI 操作，所以我们可以调用 `publishProgress()`方法并将当前的下载进度传进来，这样 `onProgressUpdate()`方法就会很快被调用，在这里就可以进行 UI 操作了。

当下载完成后，`doInBackground()`方法会返回一个布尔型变量，这样 `onPostExecute()`方法就会很快被调用，这个方法也是在主线程中运行的。然后在这里我们会根据下载的结果来弹出相应的 Toast 提示，从而完成整个 DownloadTask 任务。

简单来说，使用 AsyncTask 的诀窍就是，在 `doInBackground()`方法中执行具体的耗时任务，在 `onProgressUpdate()`方法中进行 UI 操作，在 `onPostExecute()`方法中执行一些任务的收尾工作。

如果想要启动这个任务，只需编写以下代码即可：

```
new DownloadTask().execute();
```

以上就是 AsyncTask 的基本用法，怎么样，是不是感觉简单方便了许多？我们并不需要去考虑什么异步消息处理机制，也不需要专门使用一个 Handler 来发送和接收消息，只需要调用一下 `publishProgress()`方法，就可以轻松地从子线程切换到 UI 线程了。

在本章的最佳实践环节，我们会对下载这个功能进行完整的实现。

10.3 服务的基本用法

了解了 Android 多线程编程的技术之后，下面就让我们进入到本章的正题，开始对服务的相关内容进行学习。作为 Android 四大组件之一，服务也少不了有很多非常重要的知识点，那我们自然要从最基本的用法开始学习了。

10.3.1 定义一个服务

首先看一下如何在项目中定义一个服务。新建一个 ServiceTest 项目，然后右击 com.example.servicetest→New→Service→Service，会弹出如图 10.5 所示的窗口。

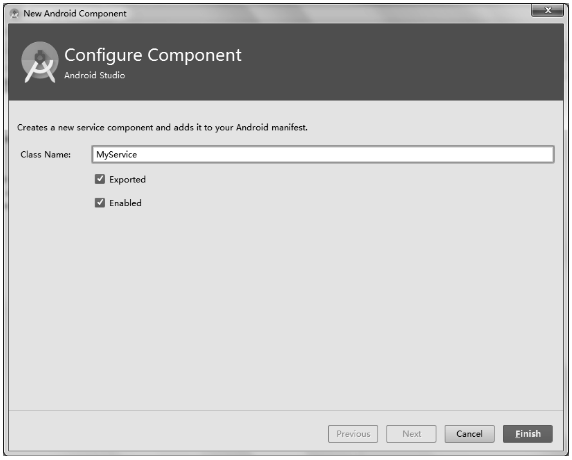


图 10.5　创建服务的窗口

可以看到，这里我们将服务命名为 MyService，Exported 属性表示是否允许除了当前程序之外的其他程序访问这个服务，Enabled 属性表示是否启用这个服务。将两个属性都勾中，点击 Finish 完成创建。

现在观察 MyService 中的代码，如下所示：

```
public class MyService extends Service {

    public MyService() {
    }

    @Override
    public IBinder onBind(Intent intent) {
        throw new UnsupportedOperationException("Not yet implemented");
    }

}
```

可以看到，MyService 是继承自 Service 类的，说明这是一个服务。目前 MyService 中可以算是空空如也，但有一个 onBind()方法特别醒目。这个方法是 Service 中唯一的一个抽象方法，所以必须要在子类里实现。我们会在后面的小节中使用到 onBind()方法，目前可以暂时将它忽略掉。

既然是定义一个服务，自然应该在服务中去处理一些事情了，那处理事情的逻辑应该写在哪里呢？这时就可以重写 Service 中的另外一些方法了，如下所示：

```
public class MyService extends Service {

    ...

    @Override
```

```
    public void onCreate() {
        super.onCreate();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
    }

}
```

可以看到，这里我们又重写了 `onCreate()`、`onStartCommand()`和 `onDestroy()`这 3 个方法，它们是每个服务中最常用到的 3 个方法了。其中 `onCreate()`方法会在服务创建的时候调用，`onStartCommand()`方法会在每次服务启动的时候调用，`onDestroy()`方法会在服务销毁的时候调用。

通常情况下，如果我们希望服务一旦启动就立刻去执行某个动作，就可以将逻辑写在 `onStartCommand()`方法里。而当服务销毁时，我们又应该在 `onDestroy()`方法中去回收那些不再使用的资源。

另外需要注意，每一个服务都需要在 AndroidManifest.xml 文件中进行注册才能生效，不知道你有没有发现，这是 Android 四大组件共有的特点。不过相信你已经猜到了，智能的 Android Studio 早已自动帮我们将这一步完成了。打开 AndroidManifest.xml 文件瞧一瞧，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.servicetest">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <service
            android:name=".MyService"
            android:enabled="true"
            android:exported="true">
        </service>
    </application>

</manifest>
```

这样的话，就已经将一个服务完全定义好了。

10.3.2　启动和停止服务

定义好了服务之后，接下来就应该考虑如何去启动以及停止这个服务。启动和停止的方法当然你也不会陌生，主要是借助 Intent 来实现的，下面就让我们在 ServiceTest 项目中尝试去启动以及停止 MyService 这个服务。

首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/start_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start Service" />

    <Button
        android:id="@+id/stop_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Stop Service" />

</LinearLayout>
```

这里我们在布局文件中加入了两个按钮，分别是用于启动服务和停止服务的。

然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button startService = (Button) findViewById(R.id.start_service);
        Button stopService = (Button) findViewById(R.id.stop_service);
        startService.setOnClickListener(this);
        stopService.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.start_service:
                Intent startIntent = new Intent(this, MyService.class);
                startService(startIntent); // 启动服务
                break;
            case R.id.stop_service:
                Intent stopIntent = new Intent(this, MyService.class);
```

```
                stopService(stopIntent); // 停止服务
                break;
            default:
                break;
        }
    }

}
```

可以看到，这里在onCreate()方法中分别获取到了Start Service按钮和Stop Service按钮的实例，并给它们注册了点击事件。然后在 Start Service 按钮的点击事件里，我们构建出了一个Intent对象，并调用startService()方法来启动MyService这个服务。在Stop Serivce按钮的点击事件里，我们同样构建出了一个Intent对象，并调用stopService()方法来停止MyService这个服务。startService()和stopService()方法都是定义在Context类中的，所以我们在活动里可以直接调用这两个方法。注意，这里完全是由活动来决定服务何时停止的，如果没有点击Stop Service按钮，服务就会一直处于运行状态。那服务有没有什么办法让自已停止下来呢？当然可以，只需要在MyService的任何一个位置调用stopSelf()方法就能让这个服务停止下来了。

那么接下来又有一个问题需要思考了，我们如何才能证实服务已经成功启动或者停止了呢？最简单的方法就是在MyService的几个方法中加入打印日志，如下所示：

```
public class MyService extends Service {

    ...

    @Override
    public void onCreate() {
        super.onCreate();
        Log.d("MyService", "onCreate executed");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d("MyService", "onStartCommand executed");
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d("MyService", "onDestroy executed");
    }

}
```

现在可以运行一下程序来进行测试了，程序的主界面如图10.6所示。



图 10.6　ServiceTest 的主界面

点击一下 Start Service 按钮，观察 logcat 中的打印日志，如图 10.7 所示。

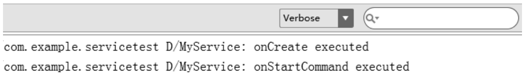


图 10.7　启动服务时的打印日志

MyService 中的 `onCreate()`和 `onStartCommand()`方法都执行了，说明这个服务确实已经启动成功了，并且你还可以在 Settings→Developer options→Running services 中找到它，如图 10.8 所示。

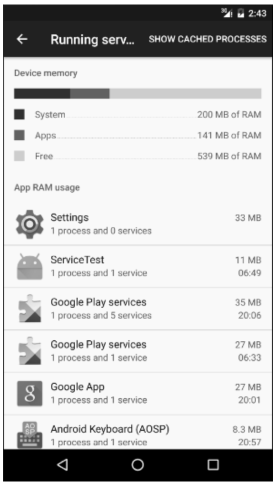


图 10.8　正在运行的服务列表

然后再点击一下 Stop Service 按钮，观察 logcat 中的打印日志，如图 10.9 所示。



图 10.9　停止服务时的打印日志

由此证明，MyService 确实已经成功停止下来了。

话说回来，虽然我们已经学会了启动服务以及停止服务的方法，不知道你心里现在有没有一个疑惑，那就是 `onCreate()`方法和 `onStartCommand()`方法到底有什么区别呢？因为刚刚点击 Start Service 按钮后两个方法都执行了。

其实 `onCreate()`方法是在服务第一次创建的时候调用的，而 `onStartCommand()`方法则在每次启动服务的时候都会调用，由于刚才我们是第一次点击 Start Service 按钮，服务此时还未创建过，所以两个方法都会执行，之后如果你再连续多点击几次 Start Service 按钮，你就会发现只有 `onStartCommand()`方法可以得到执行了。

10.3.3　活动和服务进行通信

上一小节中我们学习了启动和停止服务的方法，不知道你有没有发现，虽然服务是在活动里启动的，但在启动了服务之后，活动与服务基本就没有什么关系了。确实如此，我们在活动里调用了 `startService()`方法来启动 MyService 这个服务，然后 MyService 的 `onCreate()`和 `onStartCommand()`方法就会得到执行。之后服务会一直处于运行状态，但具体运行的是什么逻辑，活动就控制不了了。这就类似于活动通知了服务一下："你可以启动了!"然后服务就去忙自己的事情了，但活动并不知道服务到底去做了什么事情，以及完成得如何。

那么有没有什么办法能让活动和服务的关系更紧密一些呢？例如在活动中指挥服务去干什么，服务就去干什么。当然可以，这就需要借助我们刚刚忽略的 `onBind()`方法了。

比如说，目前我们希望在 MyService 里提供一个下载功能，然后在活动中可以决定何时开始下载，以及随时查看下载进度。实现这个功能的思路是创建一个专门的 `Binder` 对象来对下载功能进行管理，修改 MyService 中的代码，如下所示：

```
public class MyService extends Service {

    private DownloadBinder mBinder = new DownloadBinder();

    class DownloadBinder extends Binder {

        public void startDownload() {
            Log.d("MyService", "startDownload executed");
        }

        public int getProgress() {
            Log.d("MyService", "getProgress executed");
```

```
            return 0;
        }

    }

    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }

    ...

}
```

可以看到，这里我们新建了一个 DownloadBinder 类，并让它继承自 Binder，然后在它的内部提供了开始下载以及查看下载进度的方法。当然这只是两个模拟方法，并没有实现真正的功能，我们在这两个方法中分别打印了一行日志。

接着，在 MyService 中创建了 DownloadBinder 的实例，然后在 onBind()方法里返回了这个实例，这样 MyService 中的工作就全部完成了。

下面就要看一看，在活动中如何去调用服务里的这些方法了。首先需要在布局文件里新增两个按钮，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <Button
        android:id="@+id/bind_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Bind Service" />

    <Button
        android:id="@+id/unbind_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Unbind Service" />

</LinearLayout>
```

这两个按钮分别是用于绑定服务和取消绑定服务的，那到底谁需要去和服务绑定呢？当然就是活动了。当一个活动和服务绑定了之后，就可以调用该服务里的 Binder 提供的方法了。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private MyService.DownloadBinder downloadBinder;
```

```
    private ServiceConnection connection = new ServiceConnection() {

        @Override
        public void onServiceDisconnected(ComponentName name) {
        }

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            downloadBinder = (MyService.DownloadBinder) service;
            downloadBinder.startDownload();
            downloadBinder.getProgress();
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        Button bindService = (Button) findViewById(R.id.bind_service);
        Button unbindService = (Button) findViewById(R.id.unbind_service);
        bindService.setOnClickListener(this);
        unbindService.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            ...
            case R.id.bind_service:
                Intent bindIntent = new Intent(this, MyService.class);
                bindService(bindIntent, connection, BIND_AUTO_CREATE); // 绑定服务
                break;
            case R.id.unbind_service:
                unbindService(connection); // 解绑服务
                break;
            default:
                break;
        }
    }

}
```

这里我们首先创建了一个 ServiceConnection 的匿名类，在里面重写了 onServiceConnected()方法和 onServiceDisconnected()方法，这两个方法分别会在活动与服务成功绑定以及活动与服务的连接断开的时候调用。在 onServiceConnected()方法中，我们又通过向下转型得到了 DownloadBinder 的实例，有了这个实例，活动和服务之间的关系就变得非常紧密了。现在我们可以在活动中根据具体的场景来调用 DownloadBinder 中的任何 public 方法，即实现了指挥服务干什么服务就去干什么的功能。这里仍然只是做了个简单的测试，在 onServiceConnected()方法中调用了 DownloadBinder 的 startDownload()和 getProgress()方法。

当然，现在活动和服务其实还没进行绑定呢，这个功能是在 Bind Service 按钮的点击事件里完成的。可以看到，这里我们仍然是构建出了一个 `Intent` 对象，然后调用 `bindService()`方法将 MainActivity 和 MyService 进行绑定。`bindService()`方法接收 3 个参数，第一个参数就是刚刚构建出的 `Intent` 对象，第二个参数是前面创建出的 ServiceConnection 的实例，第三个参数则是一个标志位，这里传入 `BIND_AUTO_CREATE` 表示在活动和服务进行绑定后自动创建服务。这会使得 MyService 中的 `onCreate()`方法得到执行，但 `onStartCommand()`方法不会执行。

然后如果我们想解除活动和服务之间的绑定该怎么办呢？调用一下 `unbindService()`方法就可以了，这也是 Unbind Service 按钮的点击事件里实现的功能。

现在让我们重新运行一下程序吧，界面如图 10.10 所示。

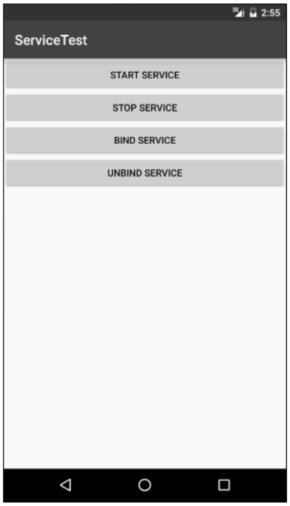


图 10.10　ServiceTest 新的主界面

点击一下 Bind Service 按钮，然后观察 logcat 中的打印日志，如图 10.11 所示。



图 10.11　绑定服务时的打印日志

可以看到，首先是 MyService 的 `onCreate()`方法得到了执行，然后 `startDownload()`和 `getProgress()`方法都得到了执行，说明我们确实已经在活动里成功调用了服务里提供的方法了。

另外需要注意，任何一个服务在整个应用程序范围内都是通用的，即 MyService 不仅可以和 MainActivity 绑定，还可以和任何一个其他的活动进行绑定，而且在绑定完成后它们都可以获取到相同的 DownloadBinder 实例。

10.4 服务的生命周期

之前我们学习过了活动以及碎片的生命周期。类似地，服务也有自己的生命周期，前面我们使用到的 onCreate()、onStartCommand()、onBind()和 onDestroy()等方法都是在服务的生命周期内可能回调的方法。

一旦在项目的任何位置调用了 Context 的 startService()方法，相应的服务就会启动起来，并回调 onStartCommand()方法。如果这个服务之前还没有创建过，onCreate()方法会先于 onStartCommand()方法执行。服务启动了之后会一直保持运行状态，直到 stopService()或 stopSelf()方法被调用。注意，虽然每调用一次 startService()方法，onStartCommand()就会执行一次，但实际上每个服务都只会存在一个实例。所以不管你调用了多少次 startService()方法，只需调用一次 stopService()或 stopSelf()方法，服务就会停止下来了。

另外，还可以调用 Context 的 bindService()来获取一个服务的持久连接，这时就会回调服务中的 onBind()方法。类似地，如果这个服务之前还没有创建过，onCreate()方法会先于 onBind()方法执行。之后，调用方可以获取到 onBind()方法里返回的 IBinder 对象的实例，这样就能自由地和服务进行通信了。只要调用方和服务之间的连接没有断开，服务就会一直保持运行状态。

当调用了 startService()方法后，又去调用 stopService()方法，这时服务中的 onDestroy()方法就会执行，表示服务已经销毁了。类似地，当调用了 bindService()方法后，又去调用 unbindService()方法，onDestroy()方法也会执行，这两种情况都很好理解。但是需要注意，我们是完全有可能对一个服务既调用了 startService()方法，又调用了 bindService()方法的，这种情况下该如何才能让服务销毁掉呢？根据 Android 系统的机制，一个服务只要被启动或者被绑定了之后，就会一直处于运行状态，必须要让以上两种条件同时不满足，服务才能被销毁。所以，这种情况下要同时调用 stopService()和 unbindService()方法，onDestroy()方法才会执行。

这样你就已经把服务的生命周期完整地走了一遍。

10.5 服务的更多技巧

以上所学的都是关于服务最基本的一些用法和概念，当然也是最常用的。不过，仅仅满足于此显然是不够的，关于服务的更多高级使用技巧还在等着我们呢，下面就赶快去看一看吧。

10.5.1 使用前台服务

服务几乎都是在后台运行的，一直以来它都是默默地做着辛苦的工作。但是服务的系统优先级还是比较低的，当系统出现内存不足的情况时，就有可能会回收掉正在后台运行的服务。如果你希望服务可以一直保持运行状态，而不会由于系统内存不足的原因导致被回收，就可以考虑使

用前台服务。前台服务和普通服务最大的区别就在于，它会一直有一个正在运行的图标在系统的状态栏显示，下拉状态栏后可以看到更加详细的信息，非常类似于通知的效果。当然有时候你也可能不仅仅是为了防止服务被回收掉才使用前台服务的，有些项目由于特殊的需求会要求必须使用前台服务，比如说彩云天气这款天气预报应用，它的服务在后台更新天气数据的同时，还会在系统状态栏一直显示当前的天气信息，如图 10.12 所示。

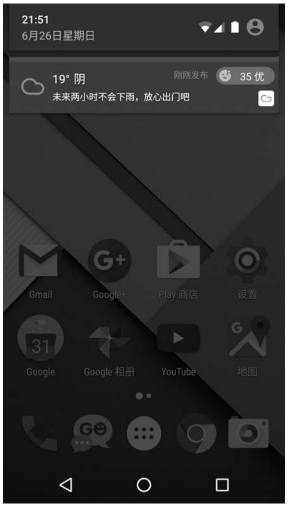


图 10.12　彩云天气的前台服务效果

那么我们就来看一下如何才能创建一个前台服务吧，其实并不复杂，修改 MyService 中的代码，如下所示：

```
public class MyService extends Service {

    ...

    @Override
    public void onCreate() {
        super.onCreate();
        Log.d("MyService", "onCreate executed");
        Intent intent = new Intent(this, MainActivity.class);
        PendingIntent pi = PendingIntent.getActivity(this, 0, intent, 0);
        Notification notification = new NotificationCompat.Builder(this)
                .setContentTitle("This is content title")
                .setContentText("This is content text")
                .setWhen(System.currentTimeMillis())
                .setSmallIcon(R.mipmap.ic_launcher)
                .setLargeIcon(BitmapFactory.decodeResource(getResources(),
                    R.mipmap.ic_launcher))
                .setContentIntent(pi)
                .build();
        startForeground(1, notification);
```

```
        }
        ...
    }
```

可以看到，这里只是修改了 onCreate()方法中的代码，相信这部分代码你会非常眼熟。没错！这就是我们在第 8 章中学习的创建通知的方法。只不过这次在构建出 Notification 对象后并没有使用 NotificationManager 来将通知显示出来，而是调用了 startForeground()方法。这个方法接收两个参数，第一个参数是通知的 id，类似于 notify()方法的第一个参数，第二个参数则是构建出的 Notification 对象。调用 startForeground()方法后就会让 MyService 变成一个前台服务，并在系统状态栏显示出来。

现在重新运行一下程序，并点击 Start Service 或 Bind Service 按钮，MyService 就会以前台服务的模式启动了，并且在系统状态栏会显示一个通知图标，下拉状态栏后可以看到该通知的详细内容，如图 10.13 所示。

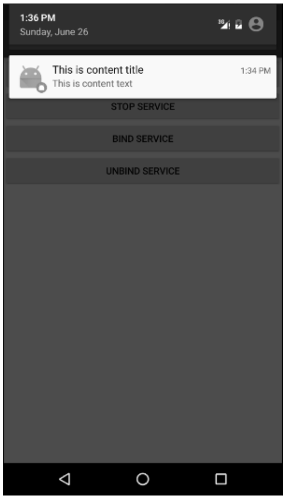


图 10.13 前台服务的状态栏效果

前台服务的用法就这么简单，只要你在第 8 章中将通知的用法掌握好了，学习本节的知识一定会特别轻松。

10.5.2 使用 IntentService

话说回来，在本章一开始的时候我们就已经知道，服务中的代码都是默认运行在主线程当中的，如果直接在服务里去处理一些耗时的逻辑，就很容易出现 ANR（Application Not Responding）的情况。

所以这个时候就需要用到 Android 多线程编程的技术了，我们应该在服务的每个具体的方法里开启一个子线程，然后在这里去处理那些耗时的逻辑。因此，一个比较标准的服务就可以写成如下形式：

```
public class MyService extends Service {

    ...

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                // 处理具体的逻辑
            }
        }).start();
        return super.onStartCommand(intent, flags, startId);
    }

}
```

但是，这种服务一旦启动之后，就会一直处于运行状态，必须调用 stopService()或者 stopSelf()方法才能让服务停止下来。所以，如果想要实现让一个服务在执行完毕后自动停止的功能，就可以这样写：

```
public class MyService extends Service {

    ...

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                // 处理具体的逻辑
                stopSelf();
            }
        }).start();
        return super.onStartCommand(intent, flags, startId);
    }

}
```

虽说这种写法并不复杂，但是总会有一些程序员忘记开启线程，或者忘记调用 stopSelf()方法。为了可以简单地创建一个异步的、会自动停止的服务，Android 专门提供了一个 IntentService 类，这个类就很好地解决了前面所提到的两种尴尬，下面我们就来看一下它的用法。

新建一个 MyIntentService 类继承自 IntentService，代码如下所示：

```
public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentService"); // 调用父类的有参构造函数
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        // 打印当前线程的 id
        Log.d("MyIntentService", "Thread id is " + Thread.currentThread(). getId());
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d("MyIntentService", "onDestroy executed");
    }

}
```

这里首先要提供一个无参的构造函数，并且必须在其内部调用父类的有参构造函数。然后要在子类中去实现 `onHandleIntent()`这个抽象方法，在这个方法中可以去处理一些具体的逻辑，而且不用担心 ANR 的问题，因为这个方法已经是在子线程中运行的了。这里为了证实一下，我们在 `onHandleIntent()`方法中打印了当前线程的 id。另外根据 `IntentService` 的特性，这个服务在运行结束后应该是会自动停止的，所以我们又重写了 `onDestroy()`方法，在这里也打印了一行日志，以证实服务是不是停止掉了。

接下来修改 activity_main.xml 中的代码，加入一个用于启动 MyIntentService 这个服务的按钮，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <Button
        android:id="@+id/start_intent_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start IntentService" />

</LinearLayout>
```

然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        Button startIntentService = (Button) findViewById(R.id.start_intent_
            service);
        startIntentService.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            ...
            case R.id.start_intent_service:
                // 打印主线程的 id
                Log.d("MainActivity", "Thread id is " + Thread.currentThread().
                    getId());
                Intent intentService = new Intent(this, MyIntentService.class);
                startService(intentService);
                break;
            default:
                break;
        }
    }

}
```

可以看到，我们在 Start IntentService 按钮的点击事件里面去启动 MyIntentService 这个服务，并在这里打印了一下主线程的 id，稍后用于和 IntentService 进行比对。你会发现，其实 IntentService 的用法和普通的服务没什么两样。

最后不要忘记，服务都是需要在 AndroidManifest.xml 里注册的，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.servicetest">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        ...

        <service android:name=".MyIntentService" />

    </application>

</manifest>
```

当然你也可以使用 Android Studio 提供的快捷方式来创建 IntentService，不过由于这样会自动生成一些我们用不到的代码，因此这里我采用了手动创建的方式。

现在重新运行一下程序，界面如图 10.14 所示。

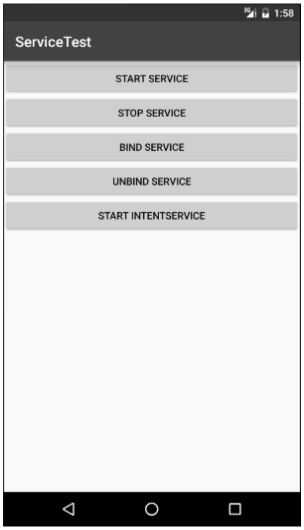


图 10.14 ServiceTest 更新后的主界面

点击 Start IntentService 按钮后，观察 logcat 中的打印日志，如图 10.15 所示。

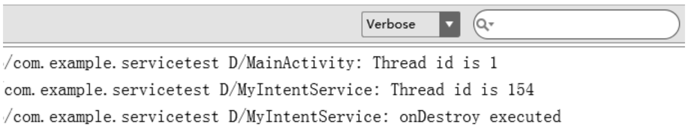


图 10.15 启动 IntentService 时的打印日志

可以看到，不仅 MyIntentService 和 MainActivity 所在的线程 id 不一样，而且 `onDestroy()` 方法也得到了执行，说明 MyIntentService 在运行完毕后确实自动停止了。集开启线程和自动停止于一身，IntentService 还是博得了不少程序员的喜爱。

好了，关于服务的知识点你已经学得够多了，下面就让我们进入到本章的最佳实践环节吧。

10.6 服务的最佳实践——完整版的下载示例

本章中你已经掌握了很多关于服务的使用技巧，但是当在真正的项目里需要用到服务的时候，可能还会有一些棘手的问题让你不知所措。因此，下面我们就来综合运用一下，尝试实现一个在服务中经常会使用到的功能——下载。

本节中我们将要编写一个完整版的下载示例，其中会涉及第 7 章、第 8 章、第 9 章和第 10 章的部分内容，算是目前为止综合程度最高的一个例子了。准备好了吗？创建一个 ServiceBestPractice 项目，然后开始本节的学习之旅吧。

首先我们需要将项目中会使用到的依赖库添加好，编辑 app/build.gradle 文件，在 dependencies 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'com.squareup.okhttp3:okhttp:3.4.1'
}
```

这里只需添加一个 OkHttp 的依赖就行了，待会儿在编写网络相关的功能时，我们将使用 OkHttp 来进行实现。

接下来需要定义一个回调接口，用于对下载过程中的各种状态进行监听和回调。新建一个 `DownloadListener` 接口，代码如下所示：

```
public interface DownloadListener {

    void onProgress(int progress);

    void onSuccess();

    void onFailed();

    void onPaused();

    void onCanceled();

}
```

可以看到，这里我们一共定义了 5 个回调方法，`onProgress()`方法用于通知当前的下载进度，`onSuccess()`方法用于通知下载成功事件，`onFailed()`方法用于通知下载失败事件，`onPaused()`方法用于通知下载暂停事件，`onCanceled()`方法用于通知下载取消事件。

回调接口定义好了之后，下面我们就可以开始编写下载功能了。这里我准备使用本章中刚学的 AsyncTask 来进行实现，新建一个 `DownloadTask` 继承自 AsyncTask，代码如下所示：

```
public class DownloadTask extends AsyncTask<String, Integer, Integer> {

    public static final int TYPE_SUCCESS = 0;
    public static final int TYPE_FAILED = 1;
    public static final int TYPE_PAUSED = 2;
    public static final int TYPE_CANCELED = 3;


    private DownloadListener listener;
```

```
private boolean isCanceled = false;

private boolean isPaused = false;

private int lastProgress;

public DownloadTask(DownloadListener listener) {
    this.listener = listener;
}

@Override
protected Integer doInBackground(String... params) {
    InputStream is = null;
    RandomAccessFile savedFile = null;
    File file = null;
    try {
        long downloadedLength = 0; // 记录已下载的文件长度
        String downloadUrl = params[0];
        String fileName = downloadUrl.substring(downloadUrl.lastIndexOf("/"));
        String directory = Environment.getExternalStoragePublicDirectory
            (Environment.DIRECTORY_DOWNLOADS).getPath();
        file = new File(directory + fileName);
        if (file.exists()) {
            downloadedLength = file.length();
        }
        long contentLength = getContentLength(downloadUrl);
        if (contentLength == 0) {
            return TYPE_FAILED;
        } else if (contentLength == downloadedLength) {
            // 已下载字节和文件总字节相等，说明已经下载完成了
            return TYPE_SUCCESS;
        }
        OkHttpClient client = new OkHttpClient();
        Request request = new Request.Builder()
                // 断点下载，指定从哪个字节开始下载
                .addHeader("RANGE", "bytes=" + downloadedLength + "-")
                .url(downloadUrl)
                .build();
        Response response = client.newCall(request).execute();
        if (response != null) {
            is = response.body().byteStream();
            savedFile = new RandomAccessFile(file, "rw");
            savedFile.seek(downloadedLength); // 跳过已下载的字节
            byte[] b = new byte[1024];
            int total = 0;
            int len;
            while ((len = is.read(b)) != -1) {
                if (isCanceled) {
                    return TYPE_CANCELED;
                } else if(isPaused) {
                    return TYPE_PAUSED;
                } else {
                    total += len;
```

```
                    savedFile.write(b, 0, len);
                    // 计算已下载的百分比
                    int progress = (int) ((total + downloadedLength) * 100 /
                        contentLength);
                    publishProgress(progress);
                }
            }
            response.body().close();
            return TYPE_SUCCESS;
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (is != null) {
                is.close();
            }
            if (savedFile != null) {
                savedFile.close();
            }
            if (isCanceled && file != null) {
                file.delete();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return TYPE_FAILED;
}

@Override
protected void onProgressUpdate(Integer... values) {
    int progress = values[0];
    if (progress > lastProgress) {
        listener.onProgress(progress);
        lastProgress = progress;
    }
}

@Override
protected void onPostExecute(Integer status) {
    switch (status) {
        case TYPE_SUCCESS:
            listener.onSuccess();
            break;
        case TYPE_FAILED:
            listener.onFailed();
            break;
        case TYPE_PAUSED:
            listener.onPaused();
            break;
        case TYPE_CANCELED:
            listener.onCanceled();
            break;
```

```
                default:
                    break;
            }
        }

        public void pauseDownload() {
            isPaused = true;
        }


        public void cancelDownload() {
            isCanceled = true;
        }

        private long getContentLength(String downloadUrl) throws IOException {
            OkHttpClient client = new OkHttpClient();
            Request request = new Request.Builder()
                    .url(downloadUrl)
                    .build();
            Response response = client.newCall(request).execute();
            if (response != null && response.isSuccessful()) {
                long contentLength = response.body().contentLength();
                response.body().close();
                return contentLength;
            }
            return 0;
        }

    }
```

这段代码就比较长了，我们需要一步步地进行分析。首先看一下 AsyncTask 中的 3 个泛型参数：第一个泛型参数指定为 `String`，表示在执行 AsyncTask 的时候需要传入一个字符串参数给后台任务；第二个泛型参数指定为 `Integer`，表示使用整型数据来作为进度显示单位；第三个泛型参数指定为 `Integer`，则表示使用整型数据来反馈执行结果。

接下来我们定义了 4 个整型常量用于表示下载的状态，`TYPE_SUCCESS` 表示下载成功，`TYPE_FAILED` 表示下载失败，`TYPE_PAUSED` 表示暂停下载，`TYPE_CANCELED` 表示取消下载。然后在 `DownloadTask` 的构造函数中要求传入一个刚刚定义的 `DownloadListener` 参数，我们待会就会将下载的状态通过这个参数进行回调。

接着就是要重写 `doInBackground()`、`onProgressUpdate()`和 `onPostExecute()`这 3 个方法了，我们之前已经学习过这 3 个方法各自的作用，因此在这里它们各自所负责的任务也是明确的：`doInBackground()`方法用于在后台执行具体的下载逻辑，`onProgressUpdate()`方法用于在界面上更新当前的下载进度，`onPostExecute()`用于通知最终的下载结果。

那么先来看一下 `doInBackground()`方法，首先我们从参数中获取到了下载的 URL 地址，并根据 URL 地址解析出了下载的文件名，然后指定将文件下载到 Environment.DIRECTORY_DOWNLOADS 目录下，也就是 SD 卡的 Download 目录。我们还要判断一下 Download 目录中是

不是已经存在要下载的文件了，如果已经存在的话则读取已下载的字节数，这样就可以在后面启用断点续传的功能。接下来先是调用了 getContentLength()方法来获取待下载文件的总长度，如果文件长度等于 0 则说明文件有问题，直接返回 TYPE_FAILED，如果文件长度等于已下载文件长度，那么就说明文件已经下载完了，直接返回 TYPE_SUCCESS 即可。紧接着使用 OkHttp 来发送一条网络请求，需要注意的是，这里在请求中添加了一个 header，用于告诉服务器我们想要从哪个字节开始下载，因为已下载过的部分就不需要再重新下载了。接下来读取服务器响应的数据，并使用 Java 的文件流方式，不断从网络上读取数据，不断写入到本地，一直到文件全部下载完成为止。在这个过程中，我们还要判断用户有没有触发暂停或者取消的操作，如果有的话则返回 TYPE_PAUSED 或 TYPE_CANCELED 来中断下载，如果没有的话则实时计算当前的下载进度，然后调用 publishProgress()方法进行通知。暂停和取消操作都是使用一个布尔型的变量来进行控制的，调用 pauseDownload()或 cancelDownload()方法即可更改变量的值。

接下来看一下 onProgressUpdate()方法，这个方法就简单得多了，它首先从参数中获取到当前的下载进度，然后和上一次的下载进度进行对比，如果有变化的话则调用 DownloadListener 的 onProgress()方法来通知下载进度更新。

最后是 onPostExecute()方法，也非常简单，就是根据参数中传入的下载状态来进行回调。下载成功就调用 DownloadListener 的 onSuccess()方法，下载失败就调用 onFailed()方法，暂停下载就调用 onPaused()方法，取消下载就调用 onCanceled()方法。

这样我们就把具体的下载功能完成了，下面为了保证 DownloadTask 可以一直在后台运行，我们还需要创建一个下载的服务。右击 com.example.servicebestpractice→New→Service→Service，新建 DownloadService，然后修改其中的代码，如下所示：

```
public class DownloadService extends Service {

    private DownloadTask downloadTask;

    private String downloadUrl;

    private DownloadListener listener = new DownloadListener() {
        @Override
        public void onProgress(int progress) {
            getNotificationManager().notify(1, getNotification("Downloading...",
                progress));
        }

        @Override
        public void onSuccess() {
            downloadTask = null;
            // 下载成功时将前台服务通知关闭，并创建一个下载成功的通知
            stopForeground(true);
            getNotificationManager().notify(1, getNotification("Download Success",
                -1));
            Toast.makeText(DownloadService.this, "Download Success",
                Toast.LENGTH_SHORT).show();
```

```
    }

    @Override
    public void onFailed() {
        downloadTask = null;
        // 下载失败时将前台服务通知关闭，并创建一个下载失败的通知
        stopForeground(true);
        getNotificationManager().notify(1, getNotification("Download Failed",
            -1));
        Toast.makeText(DownloadService.this, "Download Failed",
            Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onPaused() {
        downloadTask = null;
        Toast.makeText(DownloadService.this, "Paused", Toast.LENGTH_SHORT).
            show();
    }

    @Override
    public void onCanceled() {
        downloadTask = null;
        stopForeground(true);
        Toast.makeText(DownloadService.this, "Canceled", Toast.LENGTH_SHORT).
            show();
    }

};

private DownloadBinder mBinder = new DownloadBinder();

@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

class DownloadBinder extends Binder {

    public void startDownload(String url) {
        if (downloadTask == null) {
            downloadUrl = url;
            downloadTask = new DownloadTask(listener);
            downloadTask.execute(downloadUrl);
            startForeground(1, getNotification("Downloading...", 0));
            Toast.makeText(DownloadService.this, "Downloading...", Toast.
                LENGTH_SHORT).show();
        }
    }

    public void pauseDownload() {
        if (downloadTask != null) {
            downloadTask.pauseDownload();
        }
```

```
        }

        public void cancelDownload() {
            if (downloadTask != null) {
                downloadTask.cancelDownload();
            } else {
                if (downloadUrl != null) {
                    // 取消下载时需将文件删除，并将通知关闭
                    String fileName = downloadUrl.substring(downloadUrl.
                        lastIndexOf("/"));
                    String directory = Environment.getExternalStoragePublicDirectory
                       (Environment.DIRECTORY_DOWNLOADS).getPath();
                    File file = new File(directory + fileName);
                    if (file.exists()) {
                        file.delete();
                    }
                    getNotificationManager().cancel(1);
                    stopForeground(true);
                    Toast.makeText(DownloadService.this, "Canceled",
                        Toast.LENGTH_SHORT).show();
                }
            }
        }

    }

    private NotificationManager getNotificationManager() {
        return (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    }

    private Notification getNotification(String title, int progress) {
        Intent intent = new Intent(this, MainActivity.class);
        PendingIntent pi = PendingIntent.getActivity(this, 0, intent, 0);
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
        builder.setSmallIcon(R.mipmap.ic_launcher);
        builder.setLargeIcon(BitmapFactory.decodeResource(getResources(),
            R.mipmap.ic_launcher));
        builder.setContentIntent(pi);
        builder.setContentTitle(title);
        if (progress > = 0) {
            // 当 progress 大于或等于 0 时才需显示下载进度
            builder.setContentText(progress + "%");
            builder.setProgress(100, progress, false);
        }
        return builder.build();
    }

}
```

这段代码同样也比较长，我们还是得耐心慢慢看。首先这里创建了一个 `DownloadListener` 的匿名类实例，并在匿名类中实现了 `onProgress()`、`onSuccess()`、`onFailed()`、`onPaused()` 和 `onCanceled()`这 5 个方法。在 `onProgress()`方法中，我们调用 `getNotification()`方法构

建了一个用于显示下载进度的通知，然后调用 NotificationManager 的 notify()方法去触发这个通知，这样就可以在下拉状态栏中实时看到当前下载的进度了。在 onSuccess()方法中，我们首先是将正在下载的前台通知关闭，然后了创建一个新的通知用于告诉用户下载成功了。其他几个方法也都是类似的，分别用于告诉用户下载失败、暂停和取消这几个事件。

接下来为了要让 DownloadService 可以和活动进行通信,我们又创建了一个 DownloadBinder。DownloadBinder 中提供了 startDownload()、pauseDownload()和 cancelDownload()这 3 个方法，那么顾名思义，它们分别是用于开始下载、暂停下载和取消下载的。在 startDownload()方法中，我们创建了一个 DownloadTask 的实例，把刚才的 DownloadListener 作为参数传入，然后调用 execute()方法开启下载，并将下载文件的 URL 地址传入到 execute()方法中。同时，为了让这个下载服务成为一个前台服务，我们还调用了 startForeground()方法，这样就会在系统状态栏中创建一个持续运行的通知了。接着往下看，pauseDownload()方法中的代码就非常简单了，就是简单地调用了一下 DownloadTask 中的 pauseDownload()方法。cancelDownload()方法中的逻辑也基本类似，但是要注意，取消下载的时候我们需要将正在下载的文件删除掉，这一点和暂停下载是不同的。

另外，DownloadService 类中所有使用到的通知都是调用 getNotification()方法进行构建的，这个方法中的代码我们之前基本都是学过的，只有一个 setProgress()方法没有见过。setProgress()方法接收 3 个参数，第一个参数传入通知的最大进度，第二个参数传入通知的当前进度，第三个参数表示是否使用模糊进度条，这里传入 false。设置完 setProgress()方法，通知上就会有进度条显示出来了。

现在下载的服务也已经成功实现，后端的工作基本都完成了，那么接下来我们开始编写前端的部分。修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/start_download"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start Download" />

    <Button
        android:id="@+id/pause_download"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pause Download" />

    <Button
        android:id="@+id/cancel_download"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Cancel Download" />

</LinearLayout>
```

布局文件还是非常简单的，这里在 LinearLayout 中放置了 3 个按钮，分别用于开始下载、暂停下载和取消下载。

然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private DownloadService.DownloadBinder downloadBinder;

    private ServiceConnection connection = new ServiceConnection() {

        @Override
        public void onServiceDisconnected(ComponentName name) {
        }

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            downloadBinder = (DownloadService.DownloadBinder) service;
        }

    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button startDownload = (Button) findViewById(R.id.start_download);
        Button pauseDownload = (Button) findViewById(R.id.pause_download);
        Button cancelDownload = (Button) findViewById(R.id.cancel_download);
        startDownload.setOnClickListener(this);
        pauseDownload.setOnClickListener(this);
        cancelDownload.setOnClickListener(this);
        Intent intent = new Intent(this, DownloadService.class);
        startService(intent); // 启动服务
        bindService(intent, connection, BIND_AUTO_CREATE); // 绑定服务
        if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.
            permission.WRITE_EXTERNAL_STORAGE)!= PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(MainActivity.this, new
            String[]{ Manifest.permission. WRITE_EXTERNAL_STORAGE }, 1);
        }
    }

    @Override
    public void onClick(View v) {
        if (downloadBinder == null) {
            return;
        }
        switch (v.getId()) {
```

```
                case R.id.start_download:
                    String url = "https://raw.githubusercontent.com/guolindev/eclipse/
                        master/eclipse-inst-win64.exe";
                    downloadBinder.startDownload(url);
                    break;
                case R.id.pause_download:
                    downloadBinder.pauseDownload();
                    break;
                case R.id.cancel_download:
                    downloadBinder.cancelDownload();
                    break;
                default:
                    break;
            }
        }

        @Override
        public void onRequestPermissionsResult(int requestCode, String[] permissions,
            int[] grantResults) {
            switch (requestCode) {
                case 1:
                    if (grantResults.length > 0 && grantResults[0] != PackageManager.
                        PERMISSION_GRANTED) {
                        Toast.makeText(this, "拒绝权限将无法使用程序", Toast.LENGTH_SHORT).
                            show();
                        finish();
                    }
                    break;
                default:
            }
        }

        @Override
        protected void onDestroy() {
            super.onDestroy();
            unbindService(connection);
        }

    }
```

可以看到，这里我们首先创建了一个 ServiceConnection 的匿名类，然后在 onServiceConnected()方法中获取到 DownloadBinder 的实例，有了这个实例，我们就可以在活动中调用服务提供的各种方法了。

接下来看一下 onCreate()方法，在这里我们对各个按钮都进行了初始化操作并设置了点击事件，然后分别调用了 startService()和 bindService()方法来启动和绑定服务。这一点至关重要，因为启动服务可以保证 DownloadService 一直在后台运行，绑定服务则可以让 MainActivity 和 DownloadService 进行通信，因此两个方法调用都必不可少。在 onCreate()方法的最后，我们还进行了 WRITE_EXTERNAL_STORAGE 的运行时权限申请，因为下载文件是要下载到 SD 卡的 Download 目录下的，如果没有这个权限的话，我们整个程序都无法正常工作。

接下来的代码就非常简单了，在 onClick()方法中我们对点击事件进行判断，如果点击了开始按钮就调用 DownloadBinder 的 startDownload()方法，如果点击了暂停按钮就调用 pauseDownload()方法，如果点击了取消按钮就调用 cancelDownload()方法。startDownload()方法中你可以传入任意的下载地址，这里我使用了一个 Eclipse 的下载地址，以此向这个 Android 平台上曾经最出色的开发工具致敬。

另外还有一点需要注意，如果活动被销毁了，那么一定要记得对服务进行解绑，不然就有可能会造成内存泄漏。这里我们在 onDestroy()方法中完成了解绑操作。

现在只差最后一步了，我们还需要在 AndroidManifest.xml 文件中声明使用到的权限。当然除了权限之外，MainActivity 和 DownloadService 也是需要声明的，不过 Android Studio 应该早就帮我们将这两个组件声明好了，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.servicebestpractice">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".DownloadService"
            android:enabled="true"
            android:exported="true" />
    </application>

</manifest>
```

其中，由于我们的程序使用到了网络和访问 SD 卡的功能，因此需要声明 INTERNET 和 WRITE_EXTERNAL_STORAGE 这两个权限。

这样所有代码就都编写完了，现在终于可以运行一下程序了，如图 10.16 所示。

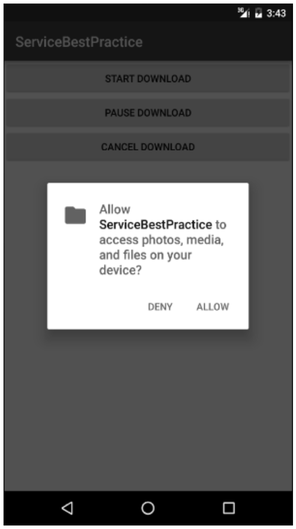


图 10.16 申请访问 SD 卡权限

程序一启动立刻就会申请访问SD卡的权限，这里我们点击ALLOW，然后点击Start Download按钮就可以开始下载了。下载过程中可以下拉系统状态栏查看实时的下载进度，如图 10.17 所示。

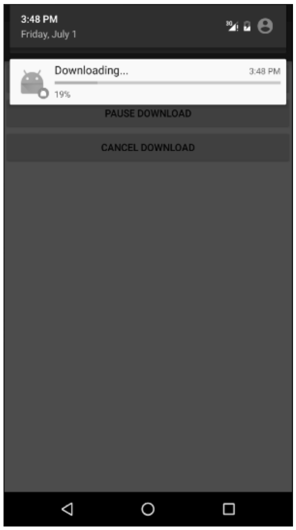


图 10.17 查看实时的下载进度

同时，我们还可以点击 Pause Download 或 Cancel Download，甚至于断网操作来测试这个下载程序的健壮性。最终下载完成后会弹出一个 Download Success 的通知，然后我们可以通过任意一个文件浏览器来查看一下 SD 卡的 Download 目录，如图 10.18 所示。



图 10.18　查看 SD 卡的 Download 目录

可以看到，文件已经成功下载下来了。

当然，我们还可以做一些更加丰富的操作，比如说再次点击 Start Download 按钮，你会发现程序会立刻弹出一个 Download Success 的提示，因为它检测到文件已经下载完成了，因而不会再重新去下载一遍。如果我们点击 Cancel Download 按钮先将下载文件删除掉，然后再点击 Start Download 按钮，你就会发现程序又会开始重新下载了。

总体来说，这个下载示例的稳定性还是挺不错的，而且综合性很强，将这个示例完全掌握了之后，你的水平肯定又更进一步了。

好了，最佳实践部分到此结束，下面我们就来回顾一下本章所学的内容吧。

10.7　小结与点评

在本章中，我们学习了很多与服务相关的重要知识点，包括 Android 多线程编程、服务的基本用法、服务的生命周期、前台服务和 IntentService 等。这些内容已经覆盖了大部分你在日常开发中可能用到的服务技术，再加上最佳实践部分学习的下载示例程序，相信以后不管遇到什么样的服务难题，你都能从容解决。

另外，本章同样是有里程碑式的纪念意义的，因为我们已经将 Android 中的四大组件全部学完，并且本书的内容也学习一大半了。对于你来说，现在你已经脱离了 Android 初级开发者的身份，并应该具备了独立完成很多功能的能力。

那么后面我们应该再接再厉，争取进一步提升自身的能力，所以现在还不是放松的时候，下一章中我们准备去学习一下 Android 特色开发的相关内容。

第 11 章

Android 特色开发——基于位置的服务

现在你已经学会了非常多的 Android 技能，并且通过这些技能你完全可以编写出相当不错的应用程序了。不过本章中，我们将要学习一些全新的 Android 技术，这些技术有别于传统的 PC 或 Web 领域的应用技术，是只有在移动设备上才能实现的。

说到只有在移动设备上才能实现的技术，很容易就让人联想到基于位置的服务（Location Based Service）。由于移动设备相比于电脑可以随身携带，我们通过地理定位的技术就可以随时得知自己所在的位置，从而围绕这一点开发出很多有意思的应用。本章中我们就将针对这一点进行讨论，学习一下基于位置的服务究竟是如何实现的。

11.1 基于位置的服务简介

基于位置的服务简称 LBS，随着移动互联网的兴起，这个技术在最近的几年里十分火爆。其实它本身并不是什么时髦的技术，主要的工作原理就是利用无线电通讯网络或 GPS 等定位方式来确定出移动设备所在的位置，而这种定位技术早在很多年前就已经出现了。

那为什么 LBS 技术直到最近几年才开始流行呢？这主要是因为，在过去移动设备的功能极其有限，即使定位到了设备所在的位置，也就仅仅只是定位到了而已，我们并不能在位置的基础上进行一些其他的操作。而现在就大大不同了，有了 Android 系统作为载体，我们可以利用定位出的位置进行许多丰富多彩的操作。比如说天气预报程序可以根据用户所在的位置自动选择城市，发微博的时候我们可以向朋友们晒一下自己在哪里，不认识路的时候随时打开地图就可以查询路线，等等。

介绍了这么多，相信你已经按捺不住了吧？我们马上就要开始本章的学习之旅，但在开始之前，还有一些事情是你必须要知道的。

首先你要清楚，基于位置的服务所围绕的核心就是要先确定出用户所在的位置。通常有两种技术方式可以实现：一种是通过 GPS 定位，一种是通过网络定位。GPS 定位的工作原理是基于手机内置的 GPS 硬件直接和卫星交互来获取当前的经纬度信息，这种定位方式精确度非常高，

但缺点是只能在室外使用，室内基本无法接收到卫星的信号。网络定位的工作原理是根据手机当前网络附近的三个基站进行测速，以此计算出手机和每个基站之间的距离，再通过三角定位确定出一个大概的位置，这种定位方式精确度一般，但优点是在室内室外都可以使用。

Android 对这两种定位方式都提供了相应的 API 支持，但是由于一些特殊原因，Google 的网络服务在中国不可访问，从而导致网络定位方式的 API 失效。而 GPS 定位虽然不需要网络，但是必须要在室外才可以使用，因此你在室内开发的时候很有可能会遇到不管使用哪种定位方式都无法成功定位的情况。

基于以上原因，我决定就不在本书中讲解 Android 原生定位 API 的用法了，而是使用一些国内第三方公司的 SDK。目前国内在这一领域做得比较好的一个是百度，一个是高德，本章我们就来学习一下百度在 LBS 方面提供的丰富多彩的功能。

11.2　申请 API Key

要想在自己的应用程序里使用百度的 LBS 功能，首先必须申请一个 API Key。你得拥有一个百度账号才能进行申请，我相信大多数人早就已经拥有了吧？如果你还没有的话，赶快去注册一个吧。

有了百度账号之后，我们就可以申请成为一名百度开发者了，登录你的百度账号，并打开 http://developer.baidu.com/user/reg 这个网址，在这里填写一些注册信息即可，如图 11.1 所示。



图 11.1　填写开发者信息

只需填写带有“*”号的那部分内容就足够了，接下来点击提交，会显示如图 11.2 所示的界面。



图 11.2　验证邮箱

接着点击“去我的邮箱”，将会进入到我们刚才填写的邮箱当中，这时收件箱中应该会有一封刚刚收到的邮件，这就是百度发送给我们的验证邮件，点击邮件当中的链接就可以完成注册了，如图 11.3 所示。



图 11.3　成为百度开发者

到此一切顺利！这样你就已经成为一名百度开发者了。接着访问 http://lbsyun.baidu.com/apiconsole/key 这个地址，然后同意百度开发者协议，会看到如图 11.4 所示的界面。



图 11.4　百度 LBS 开放平台主界面

由于这是一个刚刚注册的账号，所以目前的应用列表是空的。接下来点击创建应用就可以去申请 API Key 了，应用名称可以随便填，应用类型选择 Android SDK，启用服务保持默认即可，如图 11.5 所示。



图 11.5 创建应用界面

那么，这个发布版 SHA1 和开发版 SHA1 又是个什么东西呢？这是我们申请 API Key 所必须填写的一个字段，它指的是打包程序时所用签名文件的 SHA1 指纹，可以通过 Android Studio 查看到。打开 Android Studio 中的任意一个项目，点击右侧工具栏的 Gradle→项目名→:app→Tasks→android，如图 11.6 所示。

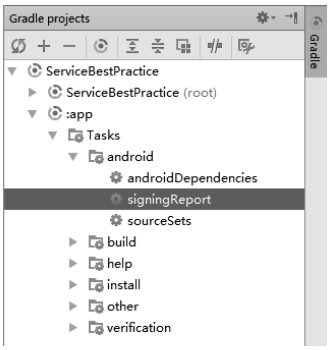


图 11.6 查看内置 Gradle Tasks

这里展示了一个 Android Studio 项目中所有内置的 Gradle Tasks，其中 signingReport 这个 Task 就可以用来查看签名文件信息。双击 signingReport，结果如图 11.7 所示。

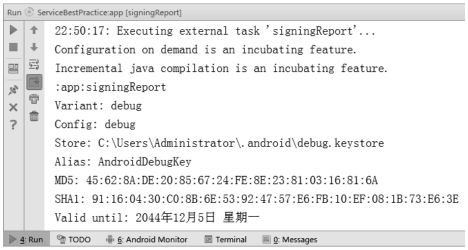


图 11.7　signingReport Task 的执行结果

其中，91:16:04:30:C0:8B:6E:53:92:47:57:E6:FB:10:EF:08:1B:73:E6:3E 就是我们所需的 SHA1 指纹了，当然你的 Android Studio 中显示的指纹和我的肯定是不一样的。另外需要注意，目前我们使用的是 debug.keystore 文件所生成的指纹，这是 Android 自动生成的一个用于测试的签名文件。而当你的应用程序发布时还需要创建一个正式的签名文件，如果要得到它的指纹，可以在 cmd 中输入如下命令：

```
keytool -list -v –keystore <签名文件路径>
```

然后输入正确的密码就可以了。创建签名文件的方法我们将在第 15 章中学习。

那么也就是说，现在得到的这个 SHA1 指纹实际上是一个开发版的 SHA1 指纹，不过因为暂时我们还没有一个发布版的 SHA1 指纹，因此这两个值都填成一样的就可以了。最后还剩下一个包名选项，虽然目前我们的应用程序还不存在，但可以先将包名预定下来，比如就叫 com.example.lbstest，这样所有的内容就都填写完整了，如图 11.8 所示。



图 11.8　填写完整所有创建应用的信息

接下来点击提交，应用就应该创建成功了，如图 11.9 所示。

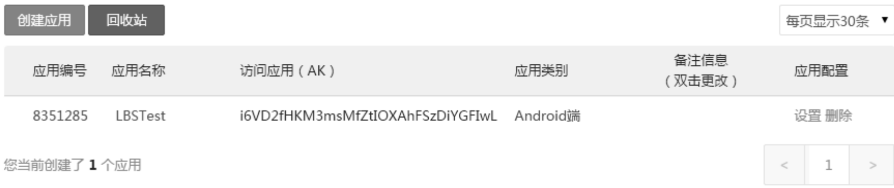


图 11.9　查看已创建的应用

其中，i6VD2fHKM3msMfZtIOXAhFSzDiYGFIwL 就是申请到的 API Key，有了它就可以进行后续的 LBS 开发工作了，那么我们马上开始吧。

11.3　使用百度定位

现在正是趁热打铁的好时机，新建一个 LBSTest 项目，包名应该就会自动被命名为 com.example.lbstest。另外需要注意，本章中所写的代码建议你都在手机上运行，虽然模拟器中也提供了模拟地理位置的功能，但在手机上可以得到真实的位置数据，你的感受会更加深刻。

11.3.1　准备 LBS SDK

在开始编码之前，我们还需要先将百度 LBS 开放平台的 SDK 准备好，下载地址是：http://lbsyun.baidu.com/sdk/download。本章中我们会用到基础地图和定位功能这两个 SDK，将它们勾选上，然后点击“开发包”下载按钮即可，如图 11.10 所示。



图 11.10　下载 SDK 界面

下载完成后对该压缩包解压，其中会有一个 libs 目录，这里面的内容就是我们所需要的一切了，如图 11.11 所示。



图 11.11　压缩包 libs 目录下的内容

libs 目录下的内容又分为两部分，BaiduLBS_Android.jar 这个文件是 Java 层要使用到的，其他子目录下的 so 文件是 Native 层要用到的。so 文件是用 C/C++语言进行编写，然后再用 NDK 编译出来的。当然这里我们并不需要去编写 C/C++的代码，因为百度都已经做好了封装，但是我们需要将 libs 目录下的每一个文件都放置到正确的位置。

首先观察一下当前的项目结构，你会发现 app 模块下面有一个 libs 目录，这里就是用来存放所有的 Jar 包的，我们将 BaiduLBS_Android.jar 复制到这里，如图 11.12 所示。

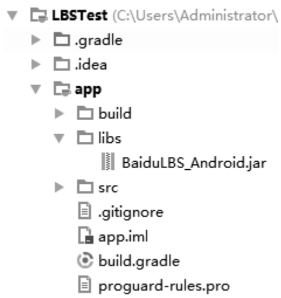


图 11.12　将 Jar 包放置到 libs 目录中

接下来展开 src/main 目录，右击该目录→New→Directory，再创建一个名为 jniLibs 的目录，这里就是专门用来存放 so 文件的，然后把压缩包里的其他所有目录直接复制到这里，如图 11.13 所示。



图 11.13　将 so 文件放置到 jniLibs 目录中

另外，虽然所有新创建的项目中，app/build.gradle 文件都会默认配置以下这段声明：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    ...
}
```

这表示会将 libs 目录下所有以.jar 结尾的文件添加到当前项目的引用中。但是由于我们是直接将 Jar 包复制到 libs 目录下的，并没有修改 gradle 文件，因此不会弹出我们平时熟悉的 Sync Now 提示。这个时候必须手动点击一下 Android Studio 顶部工具栏中的 Sync 按钮（图 11.14 中最左边的按钮），不然项目将无法引用到 Jar 包中提供的任何接口。



图 11.14　Android Studio 顶部工具栏

点击 Sync 按钮之后，libs 目录下的 jar 文件就会多出一个向右的箭头，这就表示项目已经能引用到这些 Jar 包了，如图 11.15 所示。



图 11.15　Jar 包引用成功

好了，这样我们就把 LBS 的 SDK 都准备好了，接下来开始编码吧。

11.3.2　确定自己位置的经纬度

首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/position_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

布局文件中的内容非常简单，只有一个 TextView 控件，用于稍后显示当前位置的经纬度信息。

然后修改 AndroidManifest.xml 文件中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.lbstest">

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

```
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <meta-data
            android:name="com.baidu.lbsapi.API_KEY"
            android:value="i6VD2fHKM3msMfZtIOXAhFSzDiYGFIwL" />

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service android:name="com.baidu.location.f" android:enabled="true"
            android:process=":remote">
        </service>
    </application>

</manifest>
```

AndroidManifest.xml 文件改动比较多，我们来仔细阅读一下。可以看到，这里首先添加了很多行权限声明，每一个权限都是百度 LBS SDK 内部要用到的。然后在`<application>`标签的内部添加了一个`<meta-data>`标签，这个标签的 `android:name` 部分是固定的，必须填 `com.baidu.lbsapi.API_KEY`，`android:value` 部分则应该填入我们在 11.2 节申请到的 API Key。最后，还需要再注册一个 LBS SDK 中的服务，不用对这个服务的名字感到疑惑，因为百度 LBS SDK 中的代码都是混淆过的。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    public LocationClient mLocationClient;

    private TextView positionText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mLocationClient = new LocationClient(getApplicationContext());
        mLocationClient.registerLocationListener(new MyLocationListener());
```

```
        setContentView(R.layout.activity_main);
        positionText = (TextView) findViewById(R.id.position_text_view);
        List<String> permissionList = new ArrayList<>();
        if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.
            permission.ACCESS_FINE_LOCATION)!= PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.ACCESS_FINE_LOCATION);
        }
        if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.
            permission.READ_PHONE_STATE) != PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.READ_PHONE_STATE);
        }
        if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.
            permission.WRITE_EXTERNAL_STORAGE)!= PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.WRITE_EXTERNAL_STORAGE);
        }
        if (!permissionList.isEmpty()) {
            String [] permissions = permissionList.toArray(new String[permissionList.
                size()]);
            ActivityCompat.requestPermissions(MainActivity.this, permissions, 1);
        } else {
            requestLocation();
        }
    }

    private void requestLocation() {
        mLocationClient.start();
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
        int[] grantResults) {
        switch (requestCode) {
            case 1:
                if (grantResults.length > 0) {
                    for (int result : grantResults) {
                        if (result != PackageManager.PERMISSION_GRANTED) {
                            Toast.makeText(this, "必须同意所有权限才能使用本程序",
                                Toast.LENGTH_SHORT).show();
                            finish();
                            return;
                        }
                    }
                    requestLocation();
                } else {
                    Toast.makeText(this, "发生未知错误", Toast.LENGTH_SHORT).show();
                    finish();
                }
                break;
            default:
        }
    }

    public class MyLocationListener implements BDLocationListener {

        @Override
        public void onReceiveLocation(BDLocation location) {
            runOnUiThread(new Runnable() {
```

```
                @Override
                public void run() {
                    StringBuilder currentPosition = new StringBuilder();
                    currentPosition.append("纬度：").append(location.getLatitude()).
                        append("\n");
                    currentPosition.append("经线：").append(location.getLongitude()).
                        append("\n");
                    currentPosition.append("定位方式：");
                    if (location.getLocType() == BDLocation.TypeGpsLocation) {
                        currentPosition.append("GPS");
                    } else if (location.getLocType() ==
                        BDLocation.TypeNetWorkLocation) {
                        currentPosition.append("网络");
                    }
                    positionText.setText(currentPosition);
                }
            });
        }

        @Override
        public void onConnectHotSpotMessage(String s, int i) {

        }

    }

}
```

可以看到，在 onCreate()方法中，我们首先创建了一个 LocationClient 的实例，LocationClient 的构建函数接收一个 Context 参数，这里调用 getApplicationContext()方法来获取一个全局的 Context 参数并传入。然后调用 LocationClient 的 registerLocationListener()方法来注册一个定位监听器，当获取到位置信息的时候，就会回调这个定位监听器。

接下来看一下这里运行时权限的用法，由于我们在 AndroidManifest.xml 中声明了很多权限，参考一下 7.2.1 小节中的危险权限表格可以发现，其中 ACCESS_COARSE_LOCATION、ACCESS_FINE_LOCATION、READ_PHONE_STATE、WRITE_EXTERNAL_STORAGE 这 4 个权限是需要进行运行时权限处理的，不过由于 ACCESS_COARSE_LOCATION 和 ACCESS_FINE_LOCATION 都属于同一个权限组，因此两者只要申请其一就可以了。那么怎样才能在运行时一次性申请 3 个权限呢？这里我们使用了一种新的用法，首先创建一个空的 List 集合，然后依次判断这 3 个权限有没有被授权，如果没被授权就添加到 List 集合中，最后将 List 转换成数组，再调用 ActivityCompat.requestPermissions()方法一次性申请。

除此之外，onRequestPermissionsResult()方法中对权限申请结果的逻辑处理也和之前有所不同，这次我们通过一个循环将申请的每个权限都进行了判断，如果有任何一个权限被拒绝，那么就直接调用 finish()方法关闭当前程序，只有当所有权限都被用户同意了，才会调用 requestLocation()方法开始地理位置定位。

requestLocation()方法中的代码比较简单，只是调用了一下 LocationClient 的 start()方法就能开始定位了。定位的结果会回调到我们前面注册的监听器当中，也就是

MyLocationListener。观察一下 MyLocationListener 的 onReceiveLocation()方法中，在这里我们通过 BDLocation 的 getLatitude()方法获取当前位置的纬度，通过 getLongitude()方法获取当前位置的经度，通过 getLocType()方法获取当前的定位方式，最终将结果组装成一个字符串，显示到 TextView 上面。

现在我们可以来运行一下程序了，如图 11.16 所示。毫无疑问，打开程序首先就会弹出运行时权限的申请对话框，注意看对话框的底部，提示我们一共有 3 项权限申请，当前是第 1 项，授权了第 1 项后就会显示第 2 项，这里我们全部点击允许，然后就会立刻开始定位了，结果如图 11.17 所示。



图 11.16　运行时权限申请对话框

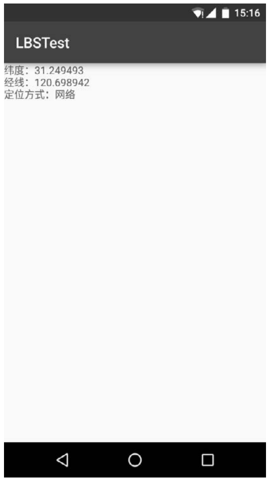


图 11.17　地理位置定位的结果

可以看到，设备当前的经纬度信息已经成功定位出来了。

不过，在默认情况下，调用 LocationClient 的 start()方法只会定位一次，如果我们正在快速移动中，怎样才能实时更新当前的位置呢？为此，百度 LBS SDK 提供了一系列的设置方法，来允许我们更改默认的行为，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    private void requestLocation() {
        initLocation();
        mLocationClient.start();
    }

    private void initLocation(){
        LocationClientOption option = new LocationClientOption();
        option.setScanSpan(5000);
        mLocationClient.setLocOption(option);
    }
```

```
    @Override
    protected void onDestroy() {
        super.onDestroy();
        mLocationClient.stop();
    }

    ...

}
```

这里增加了一个 initLocation()方法，在 initLocation()方法中我们创建了一个 LocationClientOption 对象，然后调用它的 setScanSpan()方法来设置更新的间隔。这里传入了 5000，表示每 5 秒钟会更新一下当前的位置。

最后要记得，在活动被销毁的时候一定要调用 LocationClient 的 stop()方法来停止定位，不然程序会持续在后台不停地进行定位，从而严重消耗手机的电量。

现在重新运行一下程序，然后拿着手机随处移动，你会发现界面上的经纬度信息也会跟着一起变化的。

11.3.3 选择定位模式

还记得在本章刚开始的时候说过，Android 中主要有两种定位方式吗？一种是通过 GPS 定位，一种是通过网络定位。而从上一小节中的例子中应该可以看出，我们一直是使用的网络定位。那么如何才能切换到精确度更高的 GPS 定位呢？本小节我们就来学习一下。

首先，GPS 定位功能必须要由用户主动去启用才行，不然任何应用程序都无法使用 GPS 获取到手机当前的位置信息。进入手机的设置→位置信息，如图 11.18 所示。



图 11.18 位置信息设置界面

我们可以通过顶部的开关来控制定位功能是开启还是关闭，另外，点击“模式”可以选择具体的定位模式，如图 11.19 所示。



图 11.19　选择具体的定位模式

其中，高精确度模式表示允许使用 GPS、无线网络、蓝牙或移动网络来进行定位，节电模式表示仅允许使用无线网络、蓝牙或移动网络来进行定位，而仅限设备模式表示仅允许使用 GPS 来进行定位。也就是说，如果我们想要使用 GPS 定位功能，这里必须要选择高精确度模式，或者仅限设备模式。

当然，你并不需要担心一旦启用 GPS 定位功能后，手机的电量就会直线下滑，这只是表明你已经同意让应用程序来对你的手机进行 GPS 定位了，但只有当定位操作真正开始的时候，才会影响到手机的电量。

开启了 GPS 定位功能之后，再回来看一下代码。我们可以在 `initLocation()`方法中对百度 LBS SDK 的定位模式进行指定，一共有 3 种模式可选：Hight_Accuracy、Battery_Saving 和 Device_Sensors。Hight_Accuracy 表示高精确度模式，会在 GPS 信号正常的情况下优先使用 GPS 定位，在无法接收 GPS 信号的时候使用网络定位。Battery_Saving 表示节电模式，只会使用网络进行定位。Device_Sensors 表示传感器模式，只会使用 GPS 进行定位。其中，Hight_Accuracy 是默认的模式，也就是说，我们即使不修改任何代码，只要拿着手机走到室外去，让手机可以接收到 GPS 信号，就会自动切换到 GPS 定位模式了。

当然我们也可以强制指定只使用 GPS 进行定位，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...
```

```
    private void initLocation(){
        LocationClientOption option = new LocationClientOption();
        option.setScanSpan(5000);
        option.setLocationMode(LocationClientOption.LocationMode.Device_Sensors);
        mLocationClient.setLocOption(option);
    }

    ...

}
```

这里调用了 setLocationMode()方法来将定位模式指定成传感器模式，也就是说只能使用 GPS 进行定位。重新运行一下程序，然后拿着你的手机走到室外去，结果如图 11.20 所示。

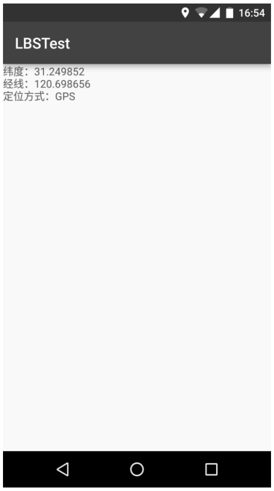


图 11.20　GPS 定位的结果

11.3.4　看得懂的位置信息

话说回来，刚才我们虽然成功获取到了设备当前位置的经纬度信息，但遗憾的是，这种经纬度的值一般人是根本看不懂的，相信谁也无法立刻答出南纬 25 度、东经 148 度是什么地方吧？为了能够更加直观地阅读，我们还需要学习一下如何获取看得懂的位置信息。

幸运的是，百度 LBS SDK 在这方面提供了非常好的支持，我们只需要进行一些简单的接口调用就能得到当前位置各种丰富的地址信息，下面就来一起看一下吧。

修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    private void initLocation(){
```

```
        LocationClientOption option = new LocationClientOption();
        option.setScanSpan(5000);
        option.setIsNeedAddress(true);
        mLocationClient.setLocOption(option);
    }

    ...

    public class MyLocationListener implements BDLocationListener {

        @Override
        public void onReceiveLocation(BDLocation location) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    StringBuilder currentPosition = new StringBuilder();
                    currentPosition.append("纬度:").append(location.getLatitude()).
                        append("\n");
                    currentPosition.append("经线:").append(location.getLongitude()).
                        append("\n");
                    currentPosition.append("国家：").append(location.getCountry()).
                        append("\n");
                    currentPosition.append("省：").append(location.getProvince()).
                        append("\n");
                    currentPosition.append("市：").append(location.getCity()).
                        append("\n");
                    currentPosition.append("区：").append(location.getDistrict()).
                        append("\n");
                    currentPosition.append("街道：").append(location.getStreet()).
                        append("\n");
                    currentPosition.append("定位方式：");
                    if (location.getLocType() == BDLocation.TypeGpsLocation) {
                        currentPosition.append("GPS");
                    } else if (location.getLocType() ==
                        BDLocation.TypeNetWorkLocation) {
                        currentPosition.append("网络");
                    }
                    positionText.setText(currentPosition);
                }
            });

        }

        ...

    }

}
```

首先在 `initLocation()`方法中,我们调用了 LocationClientOption 的 `setIsNeedAddress()`方法，并传入 `true`，这就表示我们需要获取当前位置详细的地址信息。

接下来在 MyLocationListener 的 `onReceiveLocation()`方法就可以获取到各种丰富的地址信息了，调用 `getCountry()`方法可以得到当前所在国家，调用 `getProvince()`方法可以得到当前所在省份，以此类推。另外还有一点需要注意，由于获取地址信息一定需要用到网络，因此即

使我们将定位模式指定成了 Device_Sensors，也会自动开启网络定位功能。

现在重新运行一下程序，结果如图 11.21 所示。

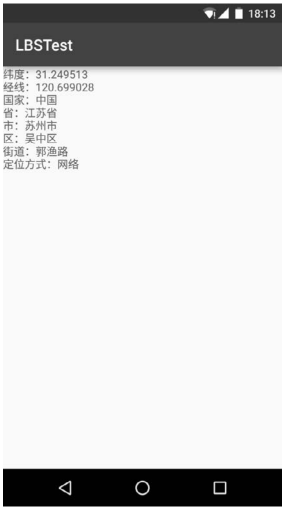


图 11.21 获取到当前位置的地址信息

可以看到，手机当前位置的地址信息已经成功显示出来了。如果你带着手机移动了较远的距离，界面上显示的位置也会跟着一起变化的。

11.4 使用百度地图

现在手机地图的应用真的可以算得上是非常广泛了，和 PC 上的地图相比，手机地图能够随时随地进行查看，并且轻松构建出行路线，使用起来明显更加地方便。但是你有没有想过，其实我们在自己的应用程序里也是可以加入地图功能的，比如优步中使用的就是百度地图。本节我们就来学习一下这方面的知识。

11.4.1 让地图显示出来

由于在上一节中我们已经将 LBS SDK 全部准备好了，其中就包括了地图功能，因此这里就不用再去下载百度地图的 SDK 了。

那么我们直接在 LBSTest 项目的基础上进行开发，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
```

```
    <TextView
        android:id="@+id/position_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone" />

    <com.baidu.mapapi.map.MapView
        android:id="@+id/bmapView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:clickable="true" />

</LinearLayout>
```

这里在布局文件中新放置了一个 MapView 控件，并让它填充满整个屏幕。这个 MapView 是由百度提供的自定义控件，所以在使用它的时候需要将完整的包名加上。另外，之前用于显示定位信息的 TextView 现在暂时用不到了，我们将它的 `visibility` 属性指定成 `gone`，让它在界面上隐藏起来。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    private MapView mapView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mLocationClient = new LocationClient(getApplicationContext());
        mLocationClient.registerLocationListener(new MyLocationListener());
        SDKInitializer.initialize(getApplicationContext());
        setContentView(R.layout.activity_main);
        mapView = (MapView) findViewById(R.id.bmapView);
        ...
    }

    ...

    @Override
    protected void onResume() {
        super.onResume();
        mapView.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        mapView.onPause();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
```

```
            mLocationClient.stop();
            mapView.onDestroy();
        }

        ...

    }
```

可以看到，这里的代码也非常简单。首先需要调用 SDKInitializer 的 `initialize()`方法来进行初始化操作，`initialize()`方法接收一个 `Context` 参数，这里我们调用 `getApplicationContext()`方法来获取一个全局的 `Context` 参数并传入。注意初始化操作一定要在 `setContentView()`方法前调用，不然的话就会出错。接下来我们调用 `findViewById()`方法获取到了 MapView 的实例，这个实例在后面的功能当中还会用到。

另外还需要重写 `onResume()`、`onPause()`和 `onDestroy()`这 3 个方法，在这里对 MapView 进行管理，以保证资源能够及时地得到释放。

好了，就是这么简单。现在重新运行一下程序，百度地图就应该成功显示出来了，如图 11.22 所示。



图 11.22　让百度地图显示出来

11.4.2　移动到我的位置

地图是成功显示出来了，但也许这并不是你想要的。因为这是一张默认的地图，显示的是北京市中心的位置，而你可能希望看到更加精细的地图信息，比如说自己所在位置的周边环境。显然，通过缩放和移动的方式来慢慢找到自己的位置是一种很愚蠢的做法。那么本小节我们就来学习一下，如何才能在地图中快速移动到自己的位置。

百度 LBS SDK 的 API 中提供了一个 BaiduMap 类，它是地图的总控制器，调用 MapView 的 getMap()方法就能获取到 BaiduMap 的实例，如下所示：

```
BaiduMap baiduMap = mapView.getMap();
```

有了 BaiduMap 后，我们就能对地图进行各种各样的操作了，比如设置地图的缩放级别以及将地图移动到某一个经纬度上。

百度地图将缩放级别的取值范围限定在 3 到 19 之间，其中小数点位的值也是可以取的，值越大，地图显示的信息就越精细。比如我们想要将缩放级别设置成 12.5，就可以这样写：

```
MapStatusUpdate update = MapStatusUpdateFactory.zoomTo(12.5f);
baiduMap.animateMapStatus(update);
```

其中 MapStatusUpdateFactory 的 zoomTo()方法接收一个 float 型的参数，就是用于设置缩放级别的，这里我们传入 12.5f。zoomTo()方法返回一个 MapStatusUpdate 对象，我们把这个对象传入 BaiduMap 的 animateMapStatus()方法当中即可完成缩放功能。

那么怎样才能让地图移动到某一个经纬度上呢？这就需要借助 LatLng 类了。其实 LatLng 并没有什么太多的用法，主要就是用于存放经纬度值的，它的构造方法接收两个参数，第一个参数是纬度值，第二个参数是经度值。之后调用 MapStatusUpdateFactory 的 newLatLng()方法将 LatLng 对象传入，newLatLng()方法返回的也是一个 MapStatusUpdate 对象，我们再把这个对象传入 BaiduMap 的 animateMapStatus()方法当中，就可以将地图移动到指定的经纬度上了，写法如下：

```
LatLng ll = new LatLng(39.915, 116.404);
MapStatusUpdate update = MapStatusUpdateFactory.newLatLng(ll);
baiduMap.animateMapStatus(update);
```

上述代码就实现了将地图移动到北纬 39.915 度、东经 116.404 度这个位置的功能。

了解了这些知识之后，接下来再去实现将地图快速移动到自己位置的功能就变得非常简单了。首先我们可以利用在 11.3 节中所学的定位技术来获得自己当前位置的经纬度，之后再按照上述的方法来将地图移动到指定的位置就可以了。

那么下面我们就来继续完善 LBSTest 这个项目，加入“移动到我的位置”这个功能。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    private BaiduMap baiduMap;

    private boolean isFirstLocate = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        mLocationClient = new LocationClient(getApplicationContext());
        mLocationClient.registerLocationListener(new MyLocationListener());
        SDKInitializer.initialize(getApplicationContext());
        setContentView(R.layout.activity_main);
        mapView = (MapView) findViewById(R.id.bmapView);
        baiduMap = mapView.getMap();
        ...
    }

    private void navigateTo(BDLocation location) {
        if (isFirstLocate) {
            LatLng ll = new LatLng(location.getLatitude(), location.getLongitude());
            MapStatusUpdate update = MapStatusUpdateFactory.newLatLng(ll);
            baiduMap.animateMapStatus(update);
            update = MapStatusUpdateFactory.zoomTo(16f);
            baiduMap.animateMapStatus(update);
            isFirstLocate = false;
        }
    }

    public class MyLocationListener implements BDLocationListener {

        @Override
        public void onReceiveLocation(BDLocation location) {
            if (location.getLocType() == BDLocation.TypeGpsLocation
                || location.getLocType() == BDLocation.TypeNetWorkLocation) {
                navigateTo(location);
            }
        }

    }

}
```

这里并没有新增多少代码，主要是加入了一个 `navigateTo()`方法。这个方法中的代码也很好理解，先是将 `BDLocation` 对象中的地理位置信息取出并封装到 `LatLng` 对象中，然后调用 MapStatusUpdateFactory 的 `newLatLng()`方法并将 `LatLng` 对象传入，接着将返回的 `MapStatusUpdate` 对象作为参数传入到 BaiduMap 的 `animateMapStatus()`方法当中，和上面介绍的用法是一模一样的。并且这里为了让地图信息可以显示得更加丰富一些，我们将缩放级别设置成了 16。另外还有一点需要注意，上述代码当中我们使用了一个 `isFirstLocate` 变量，这个变量的作用是为了防止多次调用 `animateMapStatus()`方法，因为将地图移动到我们当前的位置只需要在程序第一次定位的时候调用一次就可以了。

写好了 `navigateTo()`方法之后，剩下的事情就简单了，当定位到设备当前位置的时候，我们在 `onReceiveLocation()`方法中直接把 `BDLocation` 对象传给 `navigateTo()`方法，这样就能够让地图移动到设备所在的位置了。

现在重新运行一下程序，结果如图 11.23 所示。



图 11.23　将地图移动到设备所在的位置

11.4.3　让“我”显示在地图上

现在我们已经可以让地图显示我们周边的环境了，但是相信在你平时使用手机地图时应该会注意到，通常情况下手机地图上应该都会有一个小光标，用于显示设备当前所在的位置，并且如果设备正在移动的话，那么这个光标也会跟着一起移动。那么我们现在就继续对现有代码进行扩展，让“我”能够显示在地图上。

百度 LBS SDK 当中提供了一个 MyLocationData.Builder 类，这个类是用来封装设备当前所在位置的，我们只需将经纬度信息传入到这个类的相应方法当中就可以了，如下所示：

```
MyLocationData.Builder locationBuilder = new MyLocationData.Builder();
locationBuilder.latitude(39.915);
locationBuilder.longitude(116.404);
```

MyLocationData.Builder 类还提供了一个 build()方法，当我们把要封装的信息都设置完成之后，只需要调用它的 build()方法，就会生成一个 MyLocationData 的实例，然后再将这个实例传入到 BaiduMap 的 setMyLocationData()方法当中，就可以让设备当前的位置显示在地图上了，写法如下：

```
MyLocationData locationData = locationBuilder.build();
baiduMap.setMyLocationData(locationData);
```

大体思路就是这个样子，下面我们开始来实现一下，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mLocationClient = new LocationClient(getApplicationContext());
        mLocationClient.registerLocationListener(new MyLocationListener());
        SDKInitializer.initialize(getApplicationContext());
        setContentView(R.layout.activity_main);
        mapView = (MapView) findViewById(R.id.bmapView);
        baiduMap = mapView.getMap();
        baiduMap.setMyLocationEnabled(true);
        ...
    }

    private void navigateTo(BDLocation location) {
        if (isFirstLocate) {
            LatLng ll = new LatLng(location.getLatitude(), location.getLongitude());
            MapStatusUpdate update = MapStatusUpdateFactory.newLatLng(ll);
            baiduMap.animateMapStatus(update);
            update = MapStatusUpdateFactory.zoomTo(16f);
            baiduMap.animateMapStatus(update);
            isFirstLocate = false;
        }
        MyLocationData.Builder locationBuilder = new MyLocationData.Builder();
        locationBuilder.latitude(location.getLatitude());
        locationBuilder.longitude(location.getLongitude());
        MyLocationData locationData = locationBuilder.build();
        baiduMap.setMyLocationData(locationData);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mLocationClient.stop();
        mapView.onDestroy();
        baiduMap.setMyLocationEnabled(false);
    }

}
```

可以看到，在 `navigateTo()`方法中，我们添加了 MyLocationData 的构建逻辑，将 Location 中包含的经度和纬度分别封装到了 MyLocationData.Builder 当中，最后把 MyLocationData 设置到了 BaiduMap 的 `setMyLocationData()`方法当中。注意这段逻辑必须写在 `isFirstLocate` 这个 `if` 条件语句的外面，因为让地图移动到我们当前的位置只需要在第一次定位的时候执行，但是设备在地图上显示的位置却应该是随着设备的移动而实时改变的。

另外，根据百度地图的限制，如果我们想要使用这一功能，一定要事先调用 BaiduMap 的 `setMyLocationEnabled()`方法将此功能开启，否则设备的位置将无法在地图上显示。而在程序退出的时候，也要记得将此功能给关闭掉。

就是这么简单，现在重新运行一下程序，结果如图 11.24 所示。



图 11.24　让“我”显示在地图上

这样的话，用户就可以非常清晰地看出自己当前是在哪里了。

关于百度 LBS SDK 的用法我就准备介绍这么多，现在你已经算是成功入门了。如果想要更加深入地研究百度 LBS 的各种用法，可以到官方网站上面参考开发指南，地址是：http://lbsyun.baidu.com。另外，百度 LBS SDK 的版本未来随时都有可能更新，也许更新之后会导致书上的例子无法正常运行，因此除了照着图书学习之外，根据官网的开发指南来进行学习也是非常重要的，因为官方文档永远都是最新的。

好了，本章的主体内容到这里就结束了。下面我们将再次进入本书的特殊环节，学习一下关于 Git 的高级用法。

11.5　Git 时间——版本控制工具的高级用法

现在的你对于 Git 应该完全不会感到陌生了吧，通过了之前两节内容的学习，你已经掌握了很多 Git 中常用的命令，像提交代码这种简单的操作相信肯定是难不倒你的。

那么打开 Git Bash，并进入到 LBSTest 这个项目的根目录，然后执行提交操作：

```
git init
git add .
git commit -m "First Commit."
```

这样就将准备工作完成了，下面就让我们开始学习关于 Git 的高级用法。

11.5.1　分支的用法

分支是版本控制工具中比较高级且比较重要的一个概念，它主要的作用就是在现有代码的基础上开辟一个分叉口，使得代码可以在主干线和分支线上同时进行开发，且相互之间不会影响。分支的工作原理示意图如图 11.25 所示。

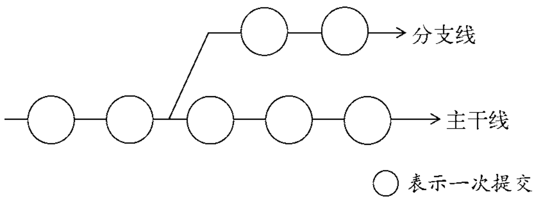


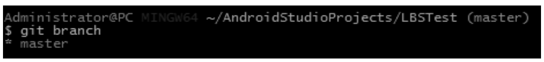
图 11.25　分支的工作原理示意图

你也许会有疑惑，为什么需要建立分支呢？只在主干线上进行开发不是挺好的吗？没错，通常情况下，只在主干线上进行开发是完全没有问题的，不过一旦涉及出版本的情况，如果不建立分支的话，你就会非常地头疼。举个简单的例子吧，比如说你们公司研发了一款不错的软件，最近刚刚完成，并推出了 1.0 版本。但是领导是不会让你们闲着的，马上提出了新的需求，让你们投入到了 1.1 版本的开发工作当中。过了几个星期，1.1 版本的功能已完成了一半，但是这个时候有用户反馈，之前上线的 1.0 版本发现了几个重大的 bug，严重影响软件的正常使用。领导也相当重视这个问题，要求你们立刻修复这些 bug，并重新发布 1.0 版本，但这个时候你就非常为难了，你会发现根本没法去修复这些 bug。因为现在 1.1 版本已开发一半了，如果在现有代码的基础上修复这些 bug，那么更新的 1.0 版本将会带有一半 1.1 版本的功能！

进退两难了是不是？但是如果你使用了分支的话，就完全不会存在这个让人头疼的问题。你只需要在发布 1.0 版本的时候建立一个分支，然后在主干线上继续开发 1.1 版本的功能。当 1.0 版本上发现任何 bug 的时候，就在分支线上进行修改，然后发布新的 1.0 版本，并记得将修改后的代码合并到主干线上。这样的话，不仅可以轻松解决掉 1.0 版本存在的 bug，而且保证了主干线上的代码也已经修复了这些 bug，当 1.1 版本发布时就不会有同样的 bug 存在了。

说了这么多，相信你也已经意识到分支的重要性了，那么我们马上来学习一下如何在 Git 中操作分支吧。

分支的英文名是 branch，如果想要查看当前的版本库当中有哪些分支，可以使用 git branch 这个命令，结果如图 11.26 所示。



```
Administrator@PC MINGW64 ~/AndroidStudioProjects/LBSTest (master)
$ git branch
* master
```

图 11.26　查看所有分支

由于目前 LBSTest 项目中还没有创建过任何分支，因此只有一个 master 分支存在，这也就是前面所说的主干线。接下来我们尝试去创建一个分支，命令如下：

```
git branch version1.0
```

这样就创建了一个名为 version1.0 的分支，我们再次输入 `git branch` 这个命令来检查一下，结果如图 11.27 所示。

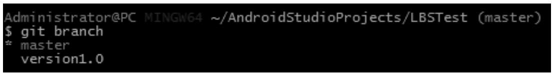


图 11.27　再次查看所有分支

可以看到，果然有一个叫作 version1.0 的分支出现了。你会发现，master 分支的前面有一个“*”号，说明目前我们的代码还是在 master 分支上的，那么怎样才能切换到 version1.0 这个分支上呢？其实也很简单，只需要使用 `checkout` 命令即可，如下所示：

```
git checkout version1.0
```

再次输入 `git branch` 来进行检查，结果如图 11.28 所示。

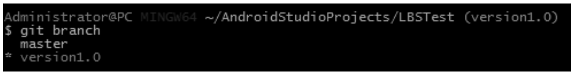


图 11.28　查看切换分支后的结果

可以看到，我们已经把代码成功切换到 version1.0 这个分支上了。

需要注意的是，在 version1.0 分支上修改并提交的代码将不会影响到 master 分支。同样的道理，在 master 分支上修改并提交的代码也不会影响到 version1.0 分支。因此，如果我们在 version1.0 分支上修复了一个 bug，在 master 分支上这个 bug 仍然是存在的。这时将修改的代码一行行复制到 master 分支上显然不是一种聪明的做法，最好的办法就是使用 `merge` 命令来完成合并操作，如下所示：

```
git checkout master
git merge version1.0
```

仅仅这样简单的两行命令，就可以把在 version1.0 分支上修改并提交的内容合并到 master 分支上了。当然，在合并分支的时候还有可能出现代码冲突的情况，这个时候你就需要静下心来慢慢地找出并解决这些冲突，Git 在这里就无法帮助你了。

最后，当我们不再需要 version1.0 这个分支的时候，可以使用如下命令将这个分支删除掉：

```
git branch -D version1.0
```

11.5.2　与远程版本库协作

可以这样说，如果你是一个人在开发，那么使用版本控制工具就远远无法发挥出它真正强大的功能。没错，所有版本控制工具最重要的一个特点就是可以使用它来进行团队合作开发。每个人的电脑上都会有一份代码，当团队的某个成员在自己的电脑上编写完成了某个功能后，就将代

码提交到服务器，其他的成员只需要将服务器上的代码同步到本地，就能保证整个团队所有人的代码都相同。这样的话，每个团队成员就可以各司其职，大家共同来完成一个较为庞大的项目。

那么如何使用 Git 来进行团队合作开发呢？这就需要有一个远程的版本库，团队的每个成员都从这个版本库中获取到最原始的代码，然后各自进行开发，并且以后每次提交的代码都同步到远程版本库上就可以了。另外，团队中的每个成员最好都要养成经常从版本库中获取最新代码的习惯，不然的话，大家的代码就很有可能经常出现冲突。

比如说现在有一个远程版本库的 Git 地址是 https://github.com/example/test.git，就可以使用如下的命令将代码下载到本地：

```
git clone https://github.com/example/test.git
```

之后你在这份代码的基础上进行了一些修改和提交，那么怎样才能把本地修改的内容同步到远程版本库上呢？这就需要借助 `push` 命令来完成了，用法如下所示：

```
git push origin master
```

其中 `origin` 部分指定的是远程版本库的 Git 地址，`master` 部分指定的是同步到哪一个分支上，上述命令就完成了将本地代码同步到 https://github.com/example/test.git 这个版本库的 `master` 分支上的功能。

知道了将本地的修改同步到远程版本库上的方法，接下来我们看一下如何将远程版本库上的修改同步到本地。Git 提供了两种命令来完成此功能，分别是 `fetch` 和 `pull`，`fetch` 的语法规则和 `push` 是差不多的，如下所示：

```
git fetch origin master
```

执行这个命令后，就会将远程版本库上的代码同步到本地，不过同步下来的代码并不会合并到任何分支上去，而是会存放到一个 `origin/master` 分支上，这时我们可以通过 `diff` 命令来查看远程版本库上到底修改了哪些东西：

```
git diff origin/master
```

之后再调用 `merge` 命令将 `origin/master` 分支上的修改合并到主分支上即可，如下所示：

```
git merge origin/master
```

而 `pull` 命令则是相当于将 `fetch` 和 `merge` 这两个命令放在一起执行了，它可以从远程版本库上获取最新的代码并且合并到本地，用法如下所示：

```
git pull origin master
```

也许你现在对远程版本库的使用还是感觉比较抽象，没关系，因为暂时我们只是了解了一下命令的用法，还没进行实践，在第 14 章当中，你将会对远程版本库的用法有更深一层的认识。

11.6　小结与点评

不得不说，本章中学到的知识应该还算是蛮有趣的吧？在这次的 Android 特色开发环节中，我们主要学习了基于位置服务的工作原理和用法，借助百度提供的 LBS SDK，我们可以随时确定自己当前位置的经纬度，并且还能获取到具体的省、市、区、街道等地址。之后又学习了百度地图的用法，不仅成功地将地图信息显示了出来，还综合利用了前面所学到的定位技术实现了一个较为完整的例子。

除了基于位置的服务之外，本章 Git 时间中继续对 Git 的用法进行了更深一步的探究，使得我们对分支和远程版本库的使用都有了一定层次的了解。

那么关于 Android 特色开发的内容就讲到这里，下一章中我们将会学习 Android 5.0 系统中新增的一套全新的知识点——Material Design。

第 12 章

最佳的 UI 体验——Material Design 实战

其实长久以来，大多数人都认为 Android 系统的 UI 并不算美观，至少没有 iOS 系统的美观。以至于很多 IT 公司在进行应用界面设计的时候，为了保证双平台的统一性，强制要求 Android 端的界面风格必须和 iOS 端一致。这种情况在现实工作当中实在是太常见了，虽然我认为这是非常不合理的。因为对于一般用户来说，他们不太可能会在两个操作系统上分别去使用同一个应用，但是却必定会在同一个操作系统上使用不同的应用。因此，同一个操作系统中各个应用之间的界面统一性要远比一个应用在双平台的界面统一性重要得多，只有这样，才能给使用者带来更好的用户体验。

但问题在于，Android 标准的界面设计风格并不是特别被大众所接受，很多公司都觉得自己完全可以设计出更加好看的界面，从而导致 Android 平台的界面风格长期难以得到统一。为了解决这个问题，谷歌也是祭出了杀手锏，在 2014 年 Google I/O 大会上重磅推出了一套全新的界面设计语言——Material Design。

本章我们就将对 Material Design 进行一次深入的学习。

12.1 什么是 Material Design

Material Design 是由谷歌的设计工程师们基于传统优秀的设计原则，结合丰富的创意和科学技术所发明的一套全新的界面设计语言，包含了视觉、运动、互动效果等特性。那么谷歌凭什么认为 Material Design 就能解决 Android 平台界面风格不统一的问题呢？一言以蔽之，好看！

没错，这次谷歌在界面设计上确实是下足了功夫，很多媒体评论，Material Design 的出现使得 Android 首次在 UI 方面超越了 iOS。按照正常的思维来想，如果各个公司都无法设计出比 Material Design 更出色的界面风格，那么它们就应该理所当然地使用 Material Design 来设计界面，从而也就能解决 Android 平台界面风格不统一的问题了。

为了做出表率，谷歌从 Android 5.0 系统开始，就将所有内置的应用都使用 Material Design 风格来进行设计。这里我随便截了两张图，你可以先欣赏一下，如图 12.1 所示。

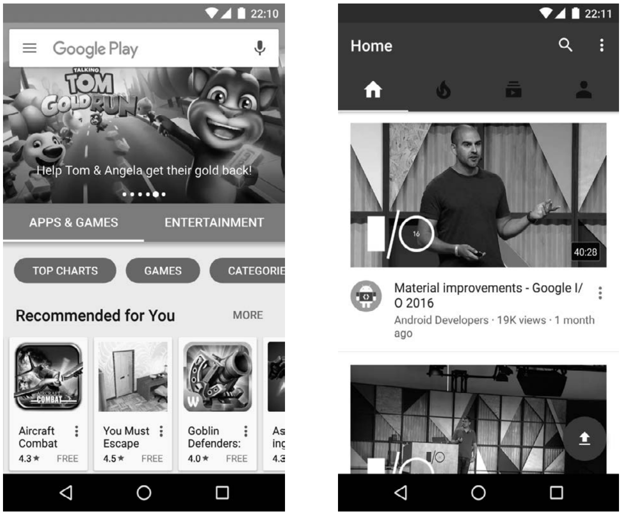


图 12.1　使用 Material Design 设计的应用

其中，左边的应用是 Play Store，右边的应用是 YouTube。可以看出，它们的界面都十分美观，而它们正是使用 Material Design 来进行设计的。

不过，在重磅推出之后，Material Design 的普及程度却不能说是特别理想。因为这只是一个推荐的设计规范，主要是面向 UI 设计人员的，而不是面向开发者的。很多开发者可能根本就搞不清楚什么样的界面和效果才叫 Material Design，就算搞清楚了，实现起来也会很费劲，因为不少 Material Design 的效果是很难实现的，而 Android 中却几乎没有提供相应的 API 支持，一切都要靠开发者自己从零写起。

谷歌当然也意识到了这个问题，于是在 2015 年的 Google I/O 大会上推出了一个 Design Support 库，这个库将 Material Design 中最具代表性的一些控件和效果进行了封装，使得开发者在即使不了解 Material Design 的情况下也能非常轻松地将自己的应用 Material 化。本章中我们就将对 Design Support 这个库进行深入的学习，并且配合一些其他的控件来完成一个优秀的 Material Design 应用。

新建一个 MaterialTest 项目，然后我们马上开始吧！

12.2　Toolbar

Toolbar 将会是我们接触的第一个 Material 控件。虽说对于 Toolbar 你暂时应该还是比较陌生的，但是对于它的另一个相关控件 ActionBar，你就应该有点熟悉了。

回忆一下，我们曾经在 3.4.1 小节为了使用一个自定义的标题栏，而把系统原生的 ActionBar 隐藏掉。没错，每个活动最顶部的那个标题栏其实就是 ActionBar，之前我们编写的所有程序里一直都有 ActionBar 的身影。

不过 ActionBar 由于其设计的原因，被限定只能位于活动的顶部，从而不能实现一些 Material Design 的效果，因此官方现在已经不再建议使用 ActionBar 了。那么本书中我也就不准备再介绍 ActionBar 的用法了，而是直接讲解现在更加推荐使用的 Toolbar。

Toolbar 的强大之处在于，它不仅继承了 ActionBar 的所有功能，而且灵活性很高，可以配合其他控件来完成一些 Material Design 的效果，下面我们就来具体学习一下。

首先你要知道，任何一个新建的项目，默认都是会显示 ActionBar 的，这个想必你已经见识过太多次了。那么这个 ActionBar 到底是从哪里来的呢？其实这是根据项目中指定的主题来显示的，打开 AndroidManifest.xml 文件看一下，如下所示：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    ...
</application>
```

可以看到，这里使用`android:theme`属性指定了一个 AppTheme 的主题。那么这个 AppTheme 又是在哪里定义的呢？打开 res/values/styles.xml 文件，代码如下所示：

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

这里定义了一个叫 AppTheme 的主题，然后指定它的 parent 主题是 Theme.AppCompat.Light.DarkActionBar。这个 DarkActionBar 是一个深色的 ActionBar 主题，我们之前所有的项目中自带的 ActionBar 就是因为指定了这个主题才出现的。

而现在我们准备使用 Toolbar 来替代 ActionBar，因此需要指定一个不带 ActionBar 的主题，通常有 Theme.AppCompat.NoActionBar 和 Theme.AppCompat.Light.NoActionBar 这两种主题可选。其中 Theme.AppCompat.NoActionBar 表示深色主题，它会将界面的主体颜色设成深色，陪衬颜色设成淡色。而 Theme.AppCompat.Light.NoActionBar 表示淡色主题，它会将界面的主体颜色设成

淡色，陪衬颜色设成深色。具体的效果你可以自己动手试一试，这里由于我们之前的程序一直都是以淡色为主的，那么我就选用淡色主题了，如下所示：

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

然后观察一下 AppTheme 中的属性重写，这里重写了 colorPrimary、colorPrimaryDark 和 colorAccent 这 3 个属性的颜色。那么这 3 个属性分别代表着什么位置的颜色呢？我用语言比较难描述清楚，还是通过一张图来理解一下吧，如图 12.2 所示。

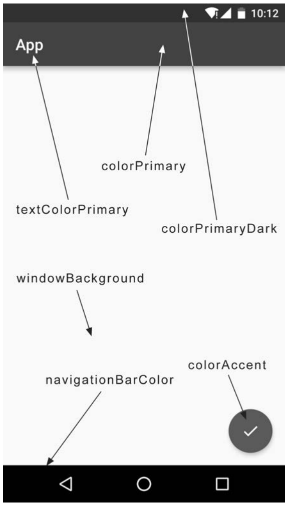


图 12.2　各属性指定颜色的位置

可以看到，每个属性所指定颜色的位置直接一目了然了。

除了上述 3 个属性之外，我们还可以通过 textColorPrimary、windowBackground 和 navigationBarColor 等属性来控制更多位置的颜色。不过唯独 colorAccent 这个属性比较难理解，它不只是用来指定这样一个按钮的颜色，而是更多表达了一个强调的意思，比如一些控件的选中状态也会使用 colorAccent 的颜色。

现在我们已经将 ActionBar 隐藏起来了，那么接下来看一看如何使用 Toolbar 来替代 ActionBar。修改 activity_main.xml 中的代码，如下所示：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

</FrameLayout>
```

虽然这段代码不长，但是里面着实有不少技术点是需要我们去仔细琢磨一下的。首先看一下第 2 行，这里使用 `xmlns:app` 指定了一个新的命名空间。思考一下，正是由于每个布局文件都会使用 `xmlns:android` 来指定一个命名空间，因此我们才能一直使用 `android:id`、`android:layout_width` 等写法，那么这里指定了 `xmlns:app`，也就是说现在可以使用 `app:attribute` 这样的写法了。但是为什么这里要指定一个 `xmlns:app` 的命名空间呢？这是由于 Material Design 是在 Android 5.0 系统中才出现的，而很多的 Material 属性在 5.0 之前的系统中并不存在，那么为了能够兼容之前的老系统，我们就不能使用 `android:attribute` 这样的写法了，而是应该使用 `app:attribute`。

接下来定义了一个 Toolbar 控件，这个控件是由 appcompat-v7 库提供的。这里我们给 Toolbar 指定了一个 id，将它的宽度设置为 `match_parent`，高度设置为 actionBar 的高度，背景色设置为 colorPrimary。不过下面的部分就稍微有点难理解了，由于我们刚才在 styles.xml 中将程序的主题指定成了淡色主题，因此 Toolbar 现在也是淡色主题，而 Toolbar 上面的各种元素就会自动使用深色系，这是为了和主体颜色区别开。但是这个效果看起来就会很差，之前使用 ActionBar 时文字都是白色的，现在变成黑色的会很难看。那么为了能让 Toolbar 单独使用深色主题，这里我们使用 `android:theme` 属性，将 Toolbar 的主题指定成了 ThemeOverlay.AppCompat.Dark.ActionBar。但是这样指定完了之后又会出现新的问题，如果 Toolbar 中有菜单按钮（我们在 2.2.5 小节中学过），那么弹出的菜单项也会变成深色主题，这样就再次变得十分难看，于是这里使用了 `app:popupTheme` 属性单独将弹出的菜单项指定成了淡色主题。之所以使用 `app:popupTheme`，是因为 `popupTheme` 这个属性是在 Android 5.0 系统中新增的，我们使用 `app:popupTheme` 的话就可以兼容 Android 5.0 以下的系统了。

如果你觉得上面的描述很绕的话，可以自己动手做一做试验，看看不指定上述主题会是什么样的效果，这样你会理解得更加深刻。

写完了布局，接下来我们修改 MainActivity，代码如下所示：

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }

}
```

这里关键的代码只有两句，首先通过 `findViewById()`得到 Toolbar 的实例，然后调用 `setSupportActionBar()`方法并将 Toolbar 的实例传入，这样我们就做到既使用了 Toolbar，又让它的外观与功能都和 ActionBar 一致了。

现在运行一下程序，效果如图 12.3 所示。

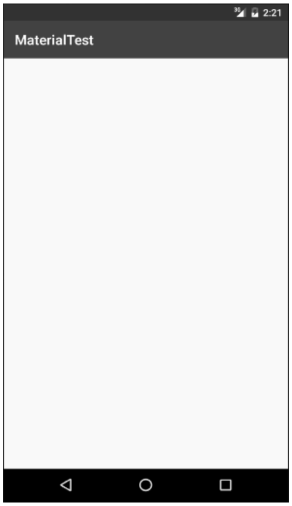


图 12.3　Toolbar 的标准界面

这个标题栏我们再熟悉不过了，虽然看上去和之前的标题栏没什么两样，但其实它已经是 Toolbar 而不是 ActionBar 了。因此它现在也具备了实现 Material Design 效果的能力，这个我们在后面就会学到。

接下来我们再学习一些 Toolbar 比较常用的功能吧，比如修改标题栏上显示的文字内容。这段文字内容是在 AndroidManifest.xml 中指定的，如下所示：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
```

```
            android:name=".MainActivity"
            android:label="Fruits">
            ...
        </activity>
    </application>
```

这里给 activity 增加了一个 android:label 属性,用于指定在 Toolbar 中显示的文字内容,如果没有指定的话，会默认使用 application 中指定的 label 内容，也就是我们的应用名称。

不过只有一个标题的 Toolbar 看起来太单调了，我们还可以再添加一些 action 按钮来让 Toolbar 更加丰富一些，这里我提前准备了几张图片来作为按钮的图标，将它们放在了 drawable-xxhdpi 目录下。现在右击 res 目录→New→Directory，创建一个 menu 文件夹。然后右击 menu 文件夹→New→Menu resource file，创建一个 toolbar.xml 文件，并编写如下代码：

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/backup"
        android:icon="@drawable/ic_backup"
        android:title="Backup"
        app:showAsAction="always" />
    <item
        android:id="@+id/delete"
        android:icon="@drawable/ic_delete"
        android:title="Delete"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/settings"
        android:icon="@drawable/ic_settings"
        android:title="Settings"
        app:showAsAction="never" />
</menu>
```

可以看到，我们通过<item>标签来定义 action 按钮，android:id 用于指定按钮的 id，android:icon 用于指定按钮的图标，android:title 用于指定按钮的文字。

接着使用 app:showAsAction 来指定按钮的显示位置,之所以这里再次使用了 app 命名空间,同样是为了能够兼容低版本的系统。showAsAction 主要有以下几种值可选：always 表示永远显示在 Toolbar 中，如果屏幕空间不够则不显示；ifRoom 表示屏幕空间足够的情况下显示在 Toolbar 中，不够的话就显示在菜单当中；never 则表示永远显示在菜单当中。注意，Toolbar 中的 action 按钮只会显示图标，菜单中的 action 按钮只会显示文字。

接下来的做法就和 2.2.5 小节中的完全一致了，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.toolbar, menu);
```

```
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.backup:
                Toast.makeText(this, "You clicked Backup", Toast.LENGTH_SHORT).
                    show();
                break;
            case R.id.delete:
                Toast.makeText(this, "You clicked Delete", Toast.LENGTH_SHORT).
                    show();
                break;
            case R.id.settings:
                Toast.makeText(this, "You clicked Settings", Toast.LENGTH_SHORT).
                    show();
                break;
            default:
        }
        return true;
    }

}
```

非常简单，我们在 onCreateOptionsMenu()方法中加载了 toolbar.xml 这个菜单文件，然后在 onOptionsItemSelected()方法中处理各个按钮的点击事件。现在重新运行一下程序，效果如图 12.4 所示。

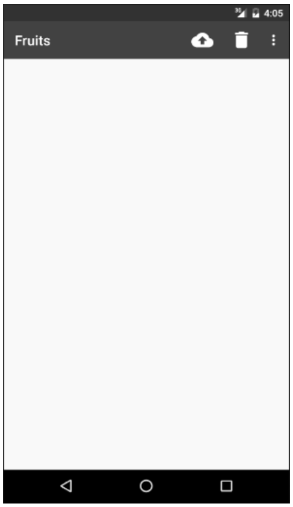


图 12.4　带有 action 按钮的 Toolbar

可以看到，Toolbar 上面现在显示了两个 action 按钮，这是因为 Backup 按钮指定的显示位置

是 always，Delete 按钮指定的显示位置是 ifRoom，而现在屏幕空间很充足，因此两个按钮都会显示在 Toolbar 中。另外一个 Settings 按钮由于指定的显示位置是 never，所以不会显示在 Toolbar 中，点击一下最右边的菜单按钮来展开菜单项，你就能找到 Settings 按钮了。另外这些 action 按钮都是可以响应点击事件的，你可以自己去试一试。

好了，关于 Toolbar 的内容就先讲这么多吧。当然 Toolbar 的功能还远远不只这些，不过我们显然无法在一节当中就把所有的用法全部学完，后面会结合其他控件来挖掘 Toolbar 的更多功能。

12.3 滑动菜单

滑动菜单可以说是 Material Design 中最常见的效果之一了，在许多著名的应用（如 Gmail、Google+等）中，都有滑动菜单的功能。虽说这个功能看上去好像挺复杂的，不过借助谷歌提供的各种工具，我们可以很轻松地实现非常炫酷的滑动菜单效果，那么我们马上开始吧。

12.3.1 DrawerLayout

所谓的滑动菜单就是将一些菜单选项隐藏起来，而不是放置在主屏幕上，然后可以通过滑动的方式将菜单显示出来。这种方式既节省了屏幕空间，又实现了非常好的动画效果，是 Material Design 中推荐的做法。

不过如果我们全靠自己去实现上述功能的话，难度恐怕就很大了。幸运的是，谷歌提供了一个 DrawerLayout 控件，借助这个控件，实现滑动菜单简单又方便。

先来简单介绍一下 DrawerLayout 的用法吧。首先它是一个布局，在布局中允许放入两个直接子控件，第一个子控件是主屏幕中显示的内容，第二个子控件是滑动菜单中显示的内容。因此，我们就可以对 activity_main.xml 中的代码做如下修改：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

```
    </FrameLayout>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:text="This is menu"
        android:textSize="30sp"
        android:background="#FFF" />

</android.support.v4.widget.DrawerLayout>
```

可以看到，这里最外层的控件使用了 DrawerLayout，这个控件是由 support-v4 库提供的。DrawerLayout 中放置了两个直接子控件，第一个子控件是 FrameLayout，用于作为主屏幕中显示的内容，当然里面还有我们刚刚定义的 Toolbar。第二个子控件这里使用了一个 TextView，用于作为滑动菜单中显示的内容，其实使用什么都可以，DrawerLayout 并没有限制只能使用固定的控件。

但是关于第二个子控件有一点需要注意，`layout_gravity` 这个属性是必须指定的，因为我们需要告诉 DrawerLayout 滑动菜单是在屏幕的左边还是右边，指定 left 表示滑动菜单在左边，指定 right 表示滑动菜单在右边。这里我指定了 start，表示会根据系统语言进行判断，如果系统语言是从左往右的，比如英语、汉语，滑动菜单就在左边，如果系统语言是从右往左的，比如阿拉伯语，滑动菜单就在右边。

没错，只需要改动这么多就可以了，现在重新运行一下程序，然后在屏幕的左侧边缘向右拖动，就可以让滑动菜单显示出来了，如图 12.5 所示。

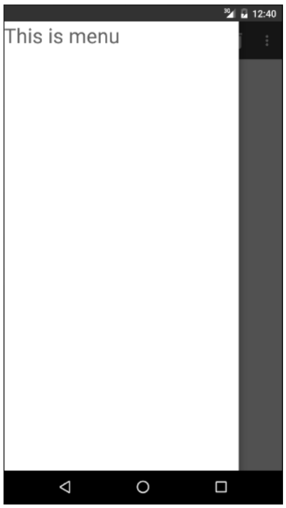


图 12.5　显示滑动菜单界面

然后向左滑动菜单，或者点击一下菜单以外的区域，都可以让滑动菜单关闭，从而回到主界面。无论是展示还是隐藏滑动菜单，都是有非常流畅的动画过渡的。

可以看到，我们只是稍微改动了一下布局文件，就能实现如此炫酷的效果，是不是觉得挺激动呢？ 不过现在的滑动菜单还有点问题，因为只有在屏幕的左侧边缘进行拖动时才能将菜单拖出来，而很多用户可能根本就不知道有这个功能，那么该怎么提示他们呢？

Material Design 建议的做法是在 Toolbar 的最左边加入一个导航按钮，点击了按钮也会将滑动菜单的内容展示出来。这样就相当于给用户提供了两种打开滑动菜单的方式，防止一些用户不知道屏幕的左侧边缘是可以拖动的。

下面我们开始来实现这个功能。首先我准备了一张导航按钮的图标 ic_menu.png，将它放在了 drawable-xxhdpi 目录下。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
            actionBar.setHomeAsUpIndicator(R.drawable.ic_menu);
        }
    }

    ...

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                mDrawerLayout.openDrawer(GravityCompat.START);
                break;
            ...
            default:
        }
        return true;
    }

}
```

这里我们并没有改动多少代码，首先调用 `findViewById()`方法得到了 DrawerLayout 的实例，然后调用 `getSupportActionBar()`方法得到了 ActionBar 的实例，虽然这个 ActionBar 的具

体实现是由 Toolbar 来完成的。接着调用 ActionBar 的 `setDisplayHomeAsUpEnabled()`方法让导航按钮显示出来，又调用了 `setHomeAsUpIndicator()`方法来设置一个导航按钮图标。实际上，Toolbar 最左侧的这个按钮就叫作 HomeAsUp 按钮，它默认的图标是一个返回的箭头，含义是返回上一个活动。很明显，这里我们将它默认的样式和作用都进行了修改。

接下来在 `onOptionsItemSelected()`方法中对 HomeAsUp 按钮的点击事件进行处理，HomeAsUp 按钮的 id 永远都是 `android.R.id.home`。然后调用 DrawerLayout 的 `openDrawer()`方法将滑动菜单展示出来，注意 `openDrawer()`方法要求传入一个 `Gravity` 参数，为了保证这里的行为和 XML 中定义的一致，我们传入了 `GravityCompat.START`。

现在重新运行一下程序，效果如图 12.6 所示。

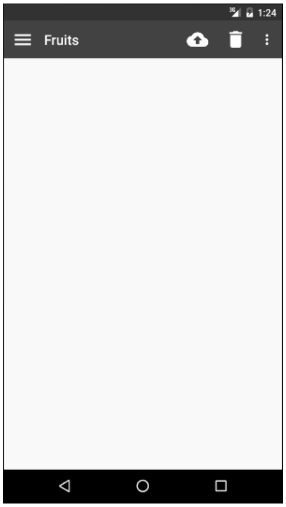


图 12.6　显示 HomeAsUp 按钮

可以看到，在 Toolbar 的最左边出现了一个导航按钮，用户看到这个按钮就知道这肯定是可以点击的。现在点击一下这个按钮，滑动菜单界面就会再次展示出来了。

12.3.2　NavigationView

目前我们已经成功实现了滑动菜单功能，其中滑动功能已经做得非常好了，但是菜单却还很丑，毕竟菜单页面仅仅使用了一个 TextView，非常单调。有对比才会有落差，我们看一下 Google+ 的滑动菜单页面是长什么样的，如图 12.7 所示。

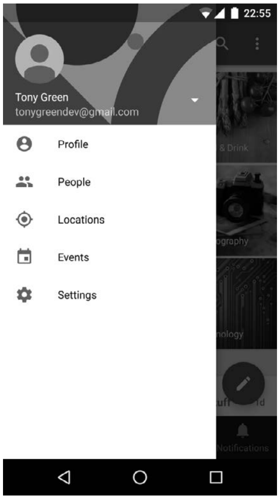


图 12.7 Google+的滑动菜单页面

经过对比之后是不是觉得我们的滑动菜单页面更丑了？不过没关系，优化滑动菜单页面，这就是我们本小节的全部目标。

事实上，你可以在滑动菜单页面定制任意的布局，不过谷歌给我们提供了一种更好的方法——使用 NavigationView。NavigationView 是 Design Support 库中提供的一个控件，它不仅是严格按照 Material Design 的要求来进行设计的，而且还可以将滑动菜单页面的实现变得非常简单。接下来我们就学习一下 NavigationView 的用法。

首先，既然这个控件是 Design Support 库中提供的，那么我们就需要将这个库引入到项目中才行。打开 app/build.gradle 文件，在 dependencies 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:design:24.2.1'
    compile 'de.hdodenhof:circleimageview:2.1.0'
}
```

这里添加了两行依赖关系，第一行就是 Design Support 库，第二行是一个开源项目 CircleImageView，它可以用来轻松实现图片圆形化的功能，我们待会就会用到它。CircleImageView 的项目主页地址是：https://github.com/hdodenhof/CircleImageView。

在开始使用 NavigationView 之前，我们还需要提前准备好两个东西：menu 和 headerLayout。menu 是用来在 NavigationView 中显示具体的菜单项的，headerLayout 则是用来在 NavigationView 中显示头部布局的。

我们先来准备 menu，这里我事先找了几张图片来作为按钮的图标，并将它们放在了

drawable-xxhdpi 目录下。然后右击 menu 文件夹→New→Menu resource file，创建一个 nav_menu.xml 文件，并编写如下代码：

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_call"
            android:icon="@drawable/nav_call"
            android:title="Call" />
        <item
            android:id="@+id/nav_friends"
            android:icon="@drawable/nav_friends"
            android:title="Friends" />
        <item
            android:id="@+id/nav_location"
            android:icon="@drawable/nav_location"
            android:title="Location" />
        <item
            android:id="@+id/nav_mail"
            android:icon="@drawable/nav_mail"
            android:title="Mail" />
        <item
            android:id="@+id/nav_task"
            android:icon="@drawable/nav_task"
            android:title="Tasks" />
    </group>
</menu>
```

我们首先在`<menu>`中嵌套了一个`<group>`标签，然后将 group 的 `checkableBehavior` 属性指定为 `single`。group 表示一个组，`checkableBehavior` 指定为 `single` 表示组中的所有菜单项只能单选。

那么下面我们来看一下这些菜单项吧。这里一共定义了 5 个 item，分别使用 `android:id` 属性指定菜单项的 id，`android:icon` 属性指定菜单项的图标，`android:title` 属性指定菜单项显示的文字。就是这么简单，现在我们已经把 menu 准备好了。

接下来应该准备 headerLayout 了，这是一个可以随意定制的布局，不过我并不想将它做得太复杂。这里简单起见，我们就在 headerLayout 中放置头像、用户名、邮箱地址这 3 项内容吧。

说到头像，那我们还需要再准备一张图片，这里我找了一张宠物图片，并把它放在了 drawable-xxhdpi 目录下。另外这张图片最好是一张正方形图片，因为待会我们会把它圆形化。然后右击 layout 文件夹→New→Layout resource file，创建一个 nav_header.xml 文件。修改其中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="180dp"
    android:padding="10dp"
    android:background="?attr/colorPrimary">
```

```
        <de.hdodenhof.circleimageview.CircleImageView
            android:id="@+id/icon_image"
            android:layout_width="70dp"
            android:layout_height="70dp"
            android:src="@drawable/nav_icon"
            android:layout_centerInParent="true" />

        <TextView
            android:id="@+id/mail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:text="tonygreendev@gmail.com"
            android:textColor="#FFF"
            android:textSize="14sp" />

        <TextView
            android:id="@+id/username"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_above="@id/mail"
            android:text="Tony Green"
            android:textColor="#FFF"
            android:textSize="14sp" />

    </RelativeLayout>
```

可以看到，布局文件的最外层是一个 RelativeLayout，我们将它的宽度设为 `match_parent`，高度设为 180dp，这是一个 NavigationView 比较适合的高度，然后指定它的背景色为 colorPrimary。

在 RelativeLayout 中我们放置了 3 个控件，CircleImageView 是一个用于将图片圆形化的控件，它的用法非常简单，基本和 ImageView 是完全一样的，这里给它指定了一张图片作为头像，然后设置为居中显示。另外两个 TextView 分别用于显示用户名和邮箱地址，它们都用到了一些 RelativeLayout 的定位属性，相信肯定难不倒你吧？

现在 menu 和 headerLayout 都准备好了，我们终于可以使用 NavigationView 了。修改 activity_main.xml 中的代码，如下所示：

```
    <android.support.v4.widget.DrawerLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:id="@+id/drawer_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
```

```
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

    </FrameLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:menu="@menu/nav_menu"
        app:headerLayout="@layout/nav_header "/>

</android.support.v4.widget.DrawerLayout>
```

可以看到，我们将之前的 TextView 换成了 NavigationView，这样滑动菜单中显示的内容也就变成 NavigationView 了。这里又通过 `app:menu` 和 `app:headerLayout` 属性将我们刚才准备好的 menu 和 headerLayout 设置了进去，这样 NavigationView 就定义完成了。

NavigationView 虽然定义完成了，但是我们还要去处理菜单项的点击事件才行。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        NavigationView  navView = (NavigationView) findViewById(R.id.nav_view);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
            actionBar.setHomeAsUpIndicator(R.drawable.ic_menu);
        }
        navView.setCheckedItem(R.id.nav_call);
        navView.setNavigationItemSelectedListener(new NavigationView.OnNavigation
            ItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(MenuItem item) {
                mDrawerLayout.closeDrawers();
                return true;
            }
        });
    }
```

```
        ...
    }
```

代码还是比较简单的，这里首先获取到了 NavigationView 的实例，然后调用它的 `setCheckedItem()`方法将 Call 菜单项设置为默认选中。接着调用了 `setNavigationItem-SelectedListener()`方法来设置一个菜单项选中事件的监听器，当用户点击了任意菜单项时，就会回调到 `onNavigationItemSelected()`方法中。我们可以在这个方法中写相应的逻辑处理，不过这里我并没有附加任何逻辑，只是调用了 DrawerLayout 的 `closeDrawers()`方法将滑动菜单关闭，这也是合情合理的做法。

现在可以重新运行一下程序了，点击一下 Toolbar 左侧的导航按钮，效果如图 12.8 所示。

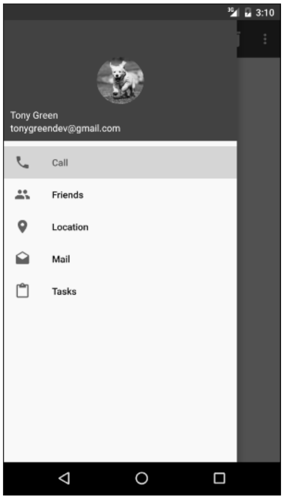


图 12.8 NavigationView 界面

怎么样？这样的滑动菜单页面，你无论如何也不能说它丑了吧？Material Design 的魅力就在这里，它真的是一种非常美观的设计理念，只要你按照它的各种规范和建议来设计界面，最终做出来的程序就是特别好看的。

相信你对现在做出来的效果也一定十分满意吧？不过不要满足于现状，后面我们会实现更加炫酷的效果。跟紧脚步，继续学习。

12.4 悬浮按钮和可交互提示

立面设计是 Material Design 中一条非常重要的设计思想，也就是说，按照 Material Design 的理念，应用程序的界面不仅仅只是一个平面，而应该是有立体效果的。在官方给出的示例中，最简单且最具代表性的立面设计就是悬浮按钮了，这种按钮不属于主界面平面的一部分，而是位于另外一个维度的，因此就会给人一种悬浮的感觉。

本节中我们会对这个悬浮按钮的效果进行学习，另外还会学习一种可交互式的提示工具。关于提示工具，我们之前一直都是使用的 Toast，但是 Toast 只能用于告知用户某某事情已经发生了，用户却不能对此做出任何的响应，那么今天我们就将在这一方面进行扩展。

12.4.1　FloatingActionButton

FloatingActionButton 是 Design Support 库中提供的一个控件，这个控件可以帮助我们比较轻松地实现悬浮按钮的效果。其实在之前的图 12.2 中，我们就已经预览过悬浮按钮是长什么样子的了，它默认会使用 colorAccent 来作为按钮的颜色，我们还可以通过给按钮指定一个图标来表明这个按钮的作用是什么。

下面开始来具体实现。首先仍然需要提前准备好一个图标，这里我放置了一张 ic_done.png 到 drawable-xxhdpi 目录下。然后修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="16dp"
            android:src="@drawable/ic_done" />
    </FrameLayout>

    ...
</android.support.v4.widget.DrawerLayout>
```

可以看到，这里我们在主屏幕布局中加入了一个 FloatingActionButton。这个控件的用法并没有什么特别的地方，layout_width 和 layout_height 属性都指定成 wrap_content，layout_gravity 属性指定将这个控件放置于屏幕的右下角，其中 end 的工作原理和之前的

start 是一样的，即如果系统语言是从左往右的，那么 end 就表示在右边，如果系统语言是从右往左的，那么 end 就表示在左边。然后通过 layout_margin 属性给控件的四周留点边距，紧贴着屏幕边缘肯定是不好看的，最后通过 src 属性给 FloatingActionButton 设置了一个图标。

没错，就是这么简单，现在我们就可以来运行一下了，效果如图 12.9 所示。

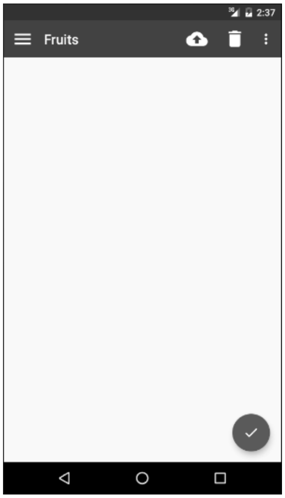


图 12.9　悬浮按钮的效果

一个漂亮的悬浮按钮就在屏幕的右下方出现了。

如果你仔细观察的话，会发现这个悬浮按钮的下面还有一点阴影。其实这很好理解，因为 FloatingActionButton 是悬浮在当前界面上的，既然是悬浮，那么就理所应当会有投影，Design Support 库连这种细节都帮我们考虑到了。

说到悬浮，其实我们还可以指定 FloatingActionButton 的悬浮高度，如下所示：

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="16dp"
    android:src="@drawable/ic_done"
    app:elevation="8dp" />
```

这里使用 app:elevation 属性来给 FloatingActionButton 指定一个高度值，高度值越大，投影范围也越大，但是投影效果越淡，高度值越小，投影范围也越小，但是投影效果越浓。当然这些效果的差异其实都不怎么明显，我个人感觉使用默认的 FloatingActionButton 效果就已经足够了。

接下来我们看一下 FloatingActionButton 是如何处理点击事件的，毕竟，一个按钮首先要能

点击才有意义。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(MainActivity.this, "FAB clicked", Toast.LENGTH_
                    SHORT).show();
            }
        });
    }

    ...

}
```

如果你在期待 FloatingActionButton 会有什么特殊用法的话，那可能就要让你失望了，它和普通的 Button 其实没什么两样，都是调用 setOnClickListener()方法来注册一个监听器，当点击按钮时，就会执行监听器中的 onClick()方法，这里我们在 onClick()方法中弹出了一个 Toast。

现在重新运行一下程序，并点击 FloatingActionButton，效果如图 12.10 所示。

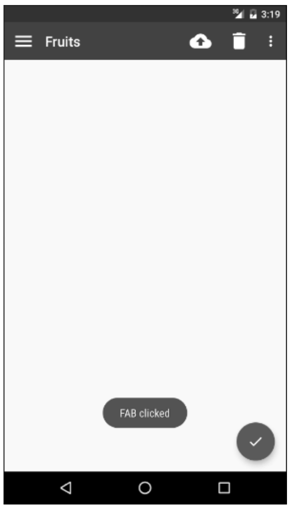


图 12.10　处理 FloatingActionButton 的点击事件

12.4.2 Snackbar

现在我们已经掌握了 FloatingActionButton 的基本用法，不过在上一小节处理点击事件的时候，仍然是使用 Toast 来作为提示工具的，本小节中我们就来学习一个 Design Support 库提供的更加先进的提示工具——Snackbar。

首先要明确，Snackbar 并不是 Toast 的替代品，它们两者之间有着不同的应用场景。Toast 的作用是告诉用户现在发生了什么事情，但同时用户只能被动接收这个事情，因为没有什么办法能让用户进行选择。而 Snackbar 则在这方面进行了扩展，它允许在提示当中加入一个可交互按钮，当用户点击按钮的时候可以执行一些额外的逻辑操作。打个比方，如果我们在执行删除操作的时候只弹出一个 Toast 提示，那么用户要是误删了某个重要数据的话肯定会十分抓狂吧，但是如果我们增加一个 Undo 按钮，就相当于给用户提供了一种弥补措施，从而大大降低了事故发生的概率，提升了用户体验。

Snackbar 的用法也非常简单，它和 Toast 是基本相似的，只不过可以额外增加一个按钮的点击事件。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Data deleted", Snackbar.LENGTH_SHORT)
                        .setAction("Undo", new View.OnClickListener() {
                            @Override
                            public void onClick(View v) {
                                Toast.makeText(MainActivity.this, "Data restored",
                                    Toast.LENGTH_SHORT).show();
                            }
                        })
                        .show();
            }
        });
    }

    ...

}
```

可以看到，这里调用了 Snackbar 的 `make()`方法来创建一个 `Snackbar` 对象，`make()`方法的

第一个参数需要传入一个 View，只要是当前界面布局的任意一个 View 都可以，Snackbar 会使用这个 View 来自动查找最外层的布局，用于展示 Snackbar。第二个参数就是 Snackbar 中显示的内容，第三个参数是 Snackbar 显示的时长。这些和 Toast 都是类似的。

接着这里又调用了一个 `setAction()`方法来设置一个动作，从而让 Snackbar 不仅仅是一个提示，而是可以和用户进行交互的。简单起见，我们在动作按钮的点击事件里面弹出一个 Toast 提示。最后调用 `show()`方法让 Snackbar 显示出来。

现在重新运行一下程序，并点击悬浮按钮，效果如图 12.11 所示。

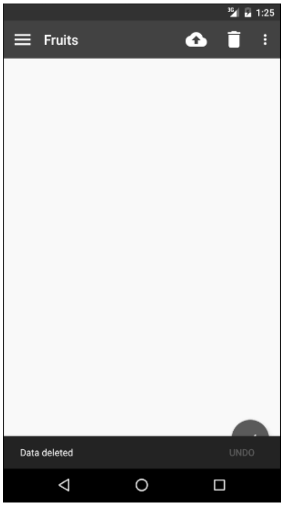


图 12.11　Snackbar 的效果

可以看到，Snackbar 从屏幕底部出现了，上面有我们所设置的提示文字，还有一个 Undo 按钮，按钮是可以点击的。过一段时间后 Snackbar 会自动从屏幕底部消失。

不管是出现还是消失，Snackbar 都是带有动画效果的，因此视觉体验也会比较好。

不过你有没有发现一个 bug，这个 Snackbar 竟然将我们的悬浮按钮给遮挡住了。虽说也不是什么重大的问题，因为 Snackbar 过一会儿就会自动消失，但这种用户体验总归是不友好的。有没有什么办法能解决一下呢？当然有，只需要借助 CoordinatorLayout 就可以轻松解决。

12.4.3　CoordinatorLayout

CoordinatorLayout 可以说是一个加强版的 FrameLayout，这个布局也是由 Design Support 库提供的。它在普通情况下的作用和 FrameLayout 基本一致，不过既然是 Design Support 库中提供的布局，那么就必然有一些 Material Design 的魔力了。

事实上，CoordinatorLayout 可以监听其所有子控件的各种事件，然后自动帮助我们做出最为

合理的响应。举个简单的例子，刚才弹出的 Snackbar 提示将悬浮按钮遮挡住了，而如果我们能让 CoordinatorLayout 监听到 Snackbar 的弹出事件，那么它会自动将内部的 FloatingActionButton 向上偏移，从而确保不会被 Snackbar 遮挡到。

至于 CoordinatorLayout 的使用也非常简单，我们只需要将原来的 FrameLayout 替换一下就可以了。修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="16dp"
            android:src="@drawable/ic_done" />
    </android.support.design.widget.CoordinatorLayout>

    ...

</android.support.v4.widget.DrawerLayout>
```

由于 CoordinatorLayout 本身就是一个加强版的 FrameLayout，因此这种替换不会有任何的副作用。现在重新运行一下程序，并点击悬浮按钮，效果如图 12.12 所示。

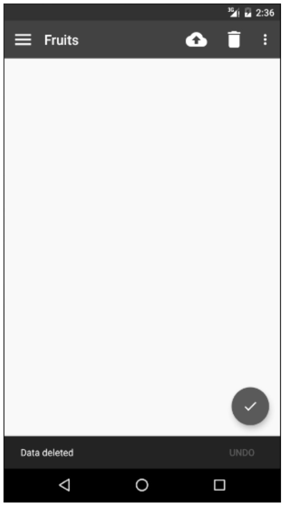


图 12.12　CoordinatorLayout 自动将悬浮按钮上移

可以看到，悬浮按钮自动向上偏移了 Snackbar 的同等高度，从而确保不会被遮挡住，当 Snackbar 消失的时候，悬浮按钮会自动向下偏移回到原来位置。

另外悬浮按钮的向上和向下偏移也是伴随着动画效果的，且和 Snackbar 完全同步，整体效果看上去特别赏心悦目。

不过我们回过头来再思考一下，刚才说的是 CoordinatorLayout 可以监听其所有子控件的各种事件，但是 Snackbar 好像并不是 CoordinatorLayout 的子控件吧，为什么它却可以被监听到呢？

其实道理很简单，还记得我们在 Snackbar 的 `make()`方法中传入的第一个参数吗？这个参数就是用来指定 Snackbar 是基于哪个 View 来触发的，刚才我们传入的是 FloatingActionButton 本身，而 FloatingActionButton 是 CoordinatorLayout 中的子控件，因此这个事件就理所应当能被监听到了。你可以自己再做个试验，如果给 Snackbar 的 `make()`方法传入一个 DrawerLayout，那么 Snackbar 就会再次遮挡住悬浮按钮，因为 DrawerLayout 不是 CoordinatorLayout 的子控件，CoordinatorLayout 也就无法监听到 Snackbar 的弹出和隐藏事件了。

本节的内容就到这里，接下来我们继续丰富 MaterialTest 项目，加入卡片式布局效果。

12.5　卡片式布局

虽然现在 MaterialTest 中已经应用了非常多的 Material Design 效果，不过你会发现，界面上最主要的一块区域还处于空白状态。这块区域通常都是用来放置应用的主体内容的，我准备使用一些精美的水果图片来填充这部分区域。

那么为了要让水果图片也能 Material 化，本节中我们将会学习如何实现卡片式布局的效果。

卡片式布局也是 Materials Design 中提出的一个新的概念，它可以让页面中的元素看起来就像在卡片中一样，并且还能拥有圆角和投影，下面我们就开始具体学习一下。

12.5.1 CardView

CardView是用于实现卡片式布局效果的重要控件，由appcompat-v7库提供。实际上，CardView也是一个 FrameLayout，只是额外提供了圆角和阴影等效果，看上去会有立体的感觉。

我们先来看一下 CardView 的基本用法吧，其实非常简单，如下所示：

```
<android.support.v7.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:cardCornerRadius="4dp"
    app:elevation="5dp">
    <TextView
        android:id="@+id/info_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</android.support.v7.widget.CardView>
```

这里定义了一个 CardView 布局，我们可以通过 `app:cardCornerRadius` 属性指定卡片圆角的弧度，数值越大，圆角的弧度也越大。另外还可以通过 `app:elevation` 属性指定卡片的高度，高度值越大，投影范围也越大，但是投影效果越淡，高度值越小，投影范围也越小，但是投影效果越浓，这一点和 FloatingActionButton 是一致的。

然后我们在 CardView 布局中放置了一个 TextView，那么这个 TextView 就会显示在一张卡片当中了，CardView 的用法就是这么简单。

但是我们显然不可能在如此宽阔的一块空白区域内只放置一张卡片，为了能够充分利用屏幕的空间，这里我准备综合运用一下第 3 章中学到的知识，使用 RecyclerView 来填充 MaterialTest 项目的主界面部分。还记得之前实现过的水果列表效果吗？这次我们将升级一下，实现一个高配版的水果列表效果。

既然是要实现水果列表，那么首先肯定需要准备许多张水果图片，这里我从网上挑选了一些精美的水果图片，将它们复制到了项目当中。

然后由于我们还需要用到 RecyclerView、CardView 这几个控件，因此必须在 app/build.gradle 文件中声明这些库的依赖才行：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:design:24.2.1'
    compile 'de.hdodenhof:circleimageview:2.1.0'
    compile 'com.android.support:recyclerview-v7:24.2.1'
    compile 'com.android.support:cardview-v7:24.2.1'
```

```
    compile 'com.github.bumptech.glide:glide:3.7.0'
}
```

注意上述声明的最后一行，这里添加了一个 Glide 库的依赖。Glide 是一个超级强大的图片加载库，它不仅可以用于加载本地图片，还可以加载网络图片、GIF 图片、甚至是本地视频。最重要的是，Glide 的用法非常简单，只需一行代码就能轻松实现复杂的图片加载功能，因此这里我们准备用它来加载水果图片。Glide 的项目主页地址是：https://github.com/bumptech/glide。

接下来开始具体的代码实现，修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

        <android.support.v7.widget.RecyclerView
            android:id="@+id/recycler_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="16dp"
            android:src="@drawable/ic_done" />
    </android.support.design.widget.CoordinatorLayout>

    ...

</android.support.v4.widget.DrawerLayout>
```

这里我们在 CoordinatorLayout 中添加了一个 RecyclerView，给它指定一个 id，然后将宽度和高度都设置为 match_parent，这样 RecyclerView 也就占满了整个布局的空间。

接着定义一个实体类 Fruit，代码如下所示：

```
public class Fruit {
```

```
    private String name;

    private int imageId;

    public Fruit(String name, int imageId) {
        this.name = name;
        this.imageId = imageId;
    }

    public String getName() {
        return name;
    }

    public int getImageId() {
        return imageId;
    }

}
```

`Fruit` 类中只有两个字段，`name` 表示水果的名字，`imageId` 表示水果对应图片的资源 id。

然后需要为RecyclerView的子项指定一个我们自定义的布局，在layout目录下新建fruit_item.xml，代码如下所示：

```
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    app:cardCornerRadius="4dp">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <ImageView
            android:id="@+id/fruit_image"
            android:layout_width="match_parent"
            android:layout_height="100dp"
            android:scaleType="centerCrop" />

        <TextView
            android:id="@+id/fruit_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_margin="5dp"
            android:textSize="16sp" />
    </LinearLayout>

</android.support.v7.widget.CardView>
```

这里使用了 CardView 来作为子项的最外层布局，从而使得 RecyclerView 中的每个元素都是在卡片当中的。CardView 由于是一个 FrameLayout，因此它没有什么方便的定位方式，这里我们只好在 CardView 中再嵌套一个 LinearLayout，然后在 LinearLayout 中放置具体的内容。

内容倒也没有什么特殊的地方，就是定义了一个 ImageView 用于显示水果的图片，又定义了一个 TextView 用于显示水果的名称，并让 TextView 在水平方向上居中显示。注意在 ImageView 中我们使用了一个 `scaleType` 属性，这个属性可以指定图片的缩放模式。由于各张水果图片的长宽比例可能都不一致，为了让所有的图片都能填充满整个 ImageView，这里使用了 centerCrop 模式，它可以让图片保持原有比例填充满 ImageView，并将超出屏幕的部分裁剪掉。

接下来需要为 RecyclerView 准备一个适配器，新建 `FruitAdapter` 类，让这个适配器继承自 RecyclerView.Adapter，并将泛型指定为 FruitAdapter.ViewHolder，代码如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {

    private Context mContext;

    private List<Fruit> mFruitList;

    static class ViewHolder extends RecyclerView.ViewHolder {
        CardView cardView;
        ImageView fruitImage;
        TextView fruitName;

        public ViewHolder(View view) {
            super(view);
            cardView = (CardView) view;
            fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
            fruitName = (TextView) view.findViewById(R.id.fruit_name);
        }
    }

    public FruitAdapter(List<Fruit> fruitList) {
        mFruitList = fruitList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        if (mContext == null) {
            mContext = parent.getContext();
        }
        View view = LayoutInflater.from(mContext).inflate(R.layout.fruit_item,
            parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Fruit fruit = mFruitList.get(position);
        holder.fruitName.setText(fruit.getName());
        Glide.with(mContext).load(fruit.getImageId()).into(holder.fruitImage);
```

```
    }

    @Override
    public int getItemCount() {
        return mFruitList.size();
    }

}
```

上述代码相信你一定很熟悉，和我们在第 3 章中编写的 FruitAdapter 几乎一模一样。唯一需要注意的是，在 `onBindViewHolder()`方法中我们使用了 Glide 来加载水果图片。

那么这里就顺便来看一下 Glide 的用法吧，其实并没有太多好讲的，因为 Glide 的用法实在是太简单了。首先调用 `Glide.with()`方法并传入一个 `Context`、`Activity` 或 `Fragment` 参数，然后调用 `load()`方法去加载图片，可以是一个 URL 地址，也可以是一个本地路径，或者是一个资源 id，最后调用 `into()`方法将图片设置到具体某一个 ImageView 中就可以了。

那么我们为什么要使用 Glide 而不是传统的设置图片方式呢？因为这次我从网上找的这些水果图片像素都非常高，如果不进行压缩就直接展示的话，很容易就会引起内存溢出。而使用 Glide 就完全不需要担心这回事，因为 Glide 在内部做了许多非常复杂的逻辑操作，其中就包括了图片压缩，我们只需要安心按照 Glide 的标准用法去加载图片就可以了。

这样我们就将 RecyclerView 的适配器也准备好了，最后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    private Fruit[] fruits = {new Fruit("Apple", R.drawable.apple), new Fruit("Banana",
        R.drawable.banana),
            new Fruit("Orange", R.drawable.orange), new Fruit("Watermelon", R.
                drawable.watermelon),
            new Fruit("Pear", R.drawable.pear), new Fruit("Grape", R.drawable.
                grape),
            new Fruit("Pineapple", R.drawable.pineapple), new Fruit("Strawberry",
                R.drawable.strawberry),
            new Fruit("Cherry", R.drawable.cherry), new Fruit("Mango", R.drawable.
                mango)};

    private List<Fruit> fruitList = new ArrayList<>();

    private FruitAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        initFruits();
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
```

```
            GridLayoutManager layoutManager = new GridLayoutManager(this, 2);
            recyclerView.setLayoutManager(layoutManager);
            adapter = new FruitAdapter(fruitList);
            recyclerView.setAdapter(adapter);
        }

        private void initFruits() {
            fruitList.clear();
            for (int i = 0; i < 50; i++) {
                Random random = new Random();
                int index = random.nextInt(fruits.length);
                fruitList.add(fruits[index]);
            }
        }

    }
```

在 MainActivity 中我们首先定义了一个数组，数组里面存放了很多个 Fruit 的实例，每个实例都代表着一种水果。然后在 initFruits()方法中，先是清空了一下 fruitList 中的数据，接着使用一个随机函数，从刚才定义的 Fruit 数组中随机挑选一个水果放入到 fruitList 当中，这样每次打开程序看到的水果数据都会是不同的。另外，为了让界面上的数据多一些，这里使用了一个循环，随机挑选 50 个水果。

之后的用法就是 RecyclerView 的标准用法了，不过这里使用了 GridLayoutManager 这种布局方式。在第 3 章中我们已经学过了 LinearLayoutManager 和 StaggeredGridLayoutManager，现在终于将所有的布局方式都补齐了。GridLayoutManager 的用法也没有什么特别之处，它的构造函数接收两个参数，第一个是 Context，第二个是列数，这里我们希望每一行中会有两列数据。

现在重新运行一下程序，效果如图 12.13 所示。

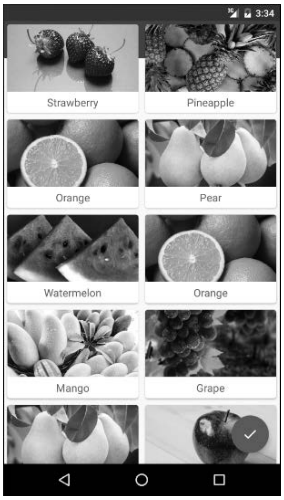


图 12.13　卡片式布局效果

可以看到，精美的水果图片成功展示出来了。每个水果都是在一张单独的卡片当中的，并且还拥有圆角和投影，是不是非常美观？另外，由于我们是使用随机的方式来获取水果数据的，因此界面上会有一些重复的水果出现，这属于正常现象。

当你陶醉于当前精美的界面的时候，你是不是忽略了一个细节？哎呀，我们的 Toolbar 怎么不见了！仔细观察一下原来是被 RecyclerView 给挡住了。这个问题又该怎么解决呢？这就需要借助到另外一个工具了——AppBarLayout。

12.5.2　AppBarLayout

首先我们来分析一下为什么 RecyclerView 会把 Toolbar 给遮挡住吧。其实并不难理解，由于 RecyclerView 和 Toolbar 都是放置在 CoordinatorLayout 中的，而前面已经说过，CoordinatorLayout 就是一个加强版的 FrameLayout，那么 FrameLayout 中的所有控件在不进行明确定位的情况下，默认都会摆放在布局的左上角，从而也就产生了遮挡的现象。其实这已经不是你第一次遇到这种情况了，我们在 3.3.3 小节学习 FrameLayout 的时候就早已见识过了控件与控件之间遮挡的效果。

既然已经找到了问题的原因，那么该如何解决呢？传统情况下，使用偏移是唯一的解决办法，即让 RecyclerView 向下偏移一个 Toolbar 的高度，从而保证不会遮挡到 Toolbar。不过我们使用的并不是普通的 FrameLayout，而是 CoordinatorLayout，因此自然会有一些更加巧妙的解决办法。

这里我准备使用 Design Support 库中提供的另外一个工具——AppBarLayout。AppBarLayout 实际上是一个垂直方向的 LinearLayout，它在内部做了很多滚动事件的封装，并应用了一些 Material Design 的设计理念。

那么我们怎样使用 AppBarLayout 才能解决前面的覆盖问题呢？其实只需要两步就可以了，第一步将 Toolbar 嵌套到 AppBarLayout 中，第二步给 RecyclerView 指定一个布局行为。修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.design.widget.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
```

```
                android:background="?attr/colorPrimary"
                android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
                app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
        </android.support.design.widget.AppBarLayout>

        <android.support.v7.widget.RecyclerView
            android:id="@+id/recycler_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="@string/appbar_scrolling_view_behavior" />

        ...
    </android.support.design.widget.CoordinatorLayout>

    ...

</android.support.v4.widget.DrawerLayout>
```

可以看到，布局文件并没有什么太大的变化。我们首先定义了一个 AppBarLayout，并将 Toolbar 放置在了 AppBarLayout 里面，然后在 RecyclerView 中使用 `app:layout_behavior` 属性指定了一个布局行为。其中 `appbar_scrolling_view_behavior` 这个字符串也是由 Design Support 库提供的。

现在重新运行一下程序，你就会发现一切都正常了，如图 12.14 所示。

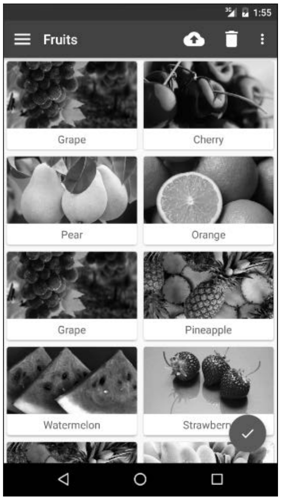


图 12.14　解决 RecyclerView 遮挡 Toolbar 的问题

虽说使用 AppBarLayout 已经成功解决了 RecyclerView 遮挡 Toolbar 的问题,但是刚才有提到过，说 AppBarLayout 中应用了一些 Material Design 的设计理念，好像从上面的例子完全体现不出来呀。事实上，当 RecyclerView 滚动的时候就已经将滚动事件都通知给 AppBarLayout 了，只是我们还没进行处理而已。那么下面就让我们来进一步优化，看看 AppBarLayout 到底能实现什

么样的 Material Design 效果。

当 AppBarLayout 接收到滚动事件的时候，它内部的子控件其实是可以指定如何去影响这些事件的，通过 app:layout_scrollFlags 属性就能实现。修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.design.widget.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:background="?attr/colorPrimary"
                android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
                app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
                app:layout_scrollFlags="scroll|enterAlways|snap" />
        </android.support.design.widget.AppBarLayout>

        ...

    </android.support.design.widget.CoordinatorLayout>

    ...

</android.support.v4.widget.DrawerLayout>
```

这里在 Toolbar 中添加了一个 app:layout_scrollFlags 属性，并将这个属性的值指定成了 scroll|enterAlways|snap。其中，scroll 表示当 RecyclerView 向上滚动的时候，Toolbar 会跟着一起向上滚动并实现隐藏；enterAlways 表示当 RecyclerView 向下滚动的时候，Toolbar 会跟着一起向下滚动并重新显示。snap 表示当 Toolbar 还没有完全隐藏或显示的时候，会根据当前滚动的距离，自动选择是隐藏还是显示。

我们要改动的就只有这一行代码而已，现在重新运行一下程序，并向上滚动 RecyclerView，效果如图 12.15 所示。



图 12.15　向上滚动 RecyclerView 隐藏 Toolbar

可以看到，随着我们向上滚动 RecyclerView，Toolbar 竟然消失了，而向下滚动 RecyclerView，Toolbar 又会重新出现。这其实也是 Material Design 中的一项重要设计思想，因为当用户在向上滚动 RecyclerView 的时候，其注意力肯定是在 RecyclerView 的内容上面的，这个时候如果 Toolbar 还占据着屏幕空间，就会在一定程度上影响用户的阅读体验，而将 Toolbar 隐藏则可以让阅读体验达到最佳状态。当用户需要操作 Toolbar 上的功能时，只需要轻微向下滚动，Toolbar 就会重新出现。这种设计方式，既保证了用户的最佳阅读效果，又不影响任何功能上的操作，Material Design 考虑得就是这么细致入微。

当然了，像这种功能，如果是使用 ActionBar 的话，那就完全不可能实现了，Toolbar 的出现为我们提供了更多的可能。

12.6　下拉刷新

下拉刷新这种功能早就不是什么新鲜的东西了，几乎所有的应用里都会有这个功能。不过市面上现有的下拉刷新功能在风格上都各不相同，并且和 Material Design 还有些格格不入的感觉。因此，谷歌为了让 Android 的下拉刷新风格能有一个统一的标准，于是在 Material Design 中制定了一个官方的设计规范。当然，我们并不需要去深入了解这个规范到底是什么样的，因为谷歌早就提供好了现成的控件，我们只需要在项目中直接使用就可以了。

SwipeRefreshLayout 就是用于实现下拉刷新功能的核心类，它是由 support-v4 库提供的。我们把想要实现下拉刷新功能的控件放置到 SwipeRefreshLayout 中，就可以迅速让这个控件支持下拉刷新。那么在 MaterialTest 项目中，应该支持下拉刷新功能的控件自然就是 RecyclerView 了。

由于 SwipeRefreshLayout 的用法也比较简单，下面我们就直接开始使用了。修改 activity_

main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        ...

        <android.support.v4.widget.SwipeRefreshLayout
            android:id="@+id/swipe_refresh"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="@string/appbar_scrolling_view_behavior">

            <android.support.v7.widget.RecyclerView
                android:id="@+id/recycler_view"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
        </android.support.v4.widget.SwipeRefreshLayout>

        ...
    </android.support.design.widget.CoordinatorLayout>

    ...

</android.support.v4.widget.DrawerLayout>
```

可以看到，这里我们在 RecyclerView 的外面又嵌套了一层 SwipeRefreshLayout，这样 RecyclerView 就自动拥有下拉刷新功能了。另外需要注意，由于 RecyclerView 现在变成了 SwipeRefreshLayout 的子控件，因此之前使用 `app:layout_behavior` 声明的布局行为现在也要移到 SwipeRefreshLayout 中才行。

不过这还没有结束，虽然 RecyclerView 已经支持下拉刷新功能了，但是我们还要在代码中处理具体的刷新逻辑才行。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    private SwipeRefreshLayout swipeRefresh;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
```

```
            swipeRefresh = (SwipeRefreshLayout) findViewById(R.id.swipe_refresh);
            swipeRefresh.setColorSchemeResources(R.color.colorPrimary);
            swipeRefresh.setOnRefreshListener(new SwipeRefreshLayout.
                OnRefreshListener() {
                @Override
                public void onRefresh() {
                    refreshFruits();
                }
            });
        }

        private void refreshFruits() {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    try {
                        Thread.sleep(2000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            initFruits();
                            adapter.notifyDataSetChanged();
                            swipeRefresh.setRefreshing(false);
                        }
                    });
                }
            }).start();
        }

        ...
    }
```

这段代码应该还是比较好理解的，首先通过 `findViewById()`方法拿到 SwipeRefreshLayout 的实例，然后调用 `setColorSchemeResources()`方法来设置下拉刷新进度条的颜色，这里我们就使用主题中的 colorPrimary 作为进度条的颜色了。接着调用 `setOnRefreshListener()`方法来设置一个下拉刷新的监听器，当触发了下拉刷新操作的时候就会回调这个监听器的 `onRefresh()`方法，然后我们在这里去处理具体的刷新逻辑就可以了。

通常情况下，`onRefresh()`方法中应该是去网络上请求最新的数据，然后再将这些数据展示出来。这里简单起见，我们就不和网络进行交互了，而是调用一个 `refreshFruits()`方法进行本地刷新操作。`refreshFruits()`方法中先是开启了一个线程，然后将线程沉睡两秒钟。之所以这么做，是因为本地刷新操作速度非常快，如果不将线程沉睡的话，刷新立刻就结束了，从而看不到刷新的过程。沉睡结束之后，这里使用了 `runOnUiThread()`方法将线程切换回主线程，然后调用 `initFruits()`方法重新生成数据，接着再调用 FruitAdapter 的 `notifyDataSetChanged()`方法通知数据发生了变化，最后调用 SwipeRefreshLayout 的 `setRefreshing()`方法并传入 `false`，用于表示刷新事件结束，并隐藏刷新进度条。

现在可以重新运行一下程序了，在屏幕的主界面向下拖动，会有一个下拉刷新的进度条出现，松手后就会自动进行刷新了，效果如图 12.16 所示。



图 12.16　实现下拉刷新效果

下拉刷新的进度条只会停留两秒钟，之后就会自动消失，界面上的水果数据也会随之更新。

这样我们就把下拉刷新的功能也成功实现了，并且这就是 Material Design 中规定的最标准的下拉刷新效果，还有什么会比这个更好看呢？目前我们的项目中已经应用了众多 Material Design 的效果，Design Support 库中的常用控件也学了大半了。不过本章的学习之旅还没有结束，在最后的尾声部分，我们再来实现一个非常震撼的 Material Design 效果——可折叠式标题栏。

12.7　可折叠式标题栏

虽说我们现在的标题栏是使用 Toolbar 来编写的，不过它看上去和传统的 ActionBar 其实没什么两样，只不过可以响应 RecyclerView 的滚动事件来进行隐藏和显示。而 Material Design 中并没有限定标题栏必须是长这个样子的，事实上，我们可以根据自己的喜好随意定制标题栏的样式。那么本节中我们就来实现一个可折叠式标题栏的效果，需要借助 CollapsingToolbarLayout 这个工具。

12.7.1　CollapsingToolbarLayout

顾名思义，CollapsingToolbarLayout 是一个作用于 Toolbar 基础之上的布局，它也是由 Design Support 库提供的。CollapsingToolbarLayout 可以让 Toolbar 的效果变得更加丰富，不仅仅是展示一个标题栏，而是能够实现非常华丽的效果。

不过，CollapsingToolbarLayout 是不能独立存在的，它在设计的时候就被限定只能作为

AppBarLayout 的直接子布局来使用。而 AppBarLayout 又必须是 CoordinatorLayout 的子布局，因此本节中我们要实现的功能其实需要综合运用前面所学的各种知识。那么话不多说，这就开始吧。

首先我们需要一个额外的活动来作为水果的详情展示界面，右击 com.example.materialtest 包→New→Activity→Empty Activity，创建一个 FruitActivity，并将布局名指定成 activity_fruit.xml，然后我们开始编写水果详情展示界面的布局。

由于整个布局文件比较复杂，这里我准备采用分段编写的方式。activity_fruit.xml 中的内容主要分为两部分，一个是水果标题栏，一个是水果内容详情，我们来一步步实现。

首先实现标题栏部分，这里使用 CoordinatorLayout 来作为最外层布局，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</android.support.design.widget.CoordinatorLayout>
```

一开始的代码还是比较简单的，相信没有什么需要解释的地方。注意始终记得要定义一个 xmlns:app 的命名空间，在 Material Design 的开发中会经常用到它。

接着我们在 CoordinatorLayout 中嵌套一个 AppBarLayout，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">
    </android.support.design.widget.AppBarLayout>

</android.support.design.widget.CoordinatorLayout>
```

目前为止也没有什么难理解的地方，我们给 AppBarLayout 定义了一个 id，将它的宽度指定为 match_parent，高度指定为 250dp。当然这里的高度值你可以随意指定，不过我尝试之后发现 250dp 的视觉效果比较好。

接下来我们在 AppBarLayout 中再嵌套一个 CollapsingToolbarLayout，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:contentScrim="?attr/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed">
        </android.support.design.widget.CollapsingToolbarLayout>

    </android.support.design.widget.AppBarLayout>

</android.support.design.widget.CoordinatorLayout>
```

从现在开始就稍微有点难理解了，这里我们使用了新的布局 CollapsingToolbarLayout。其中，`id`、`layout_width` 和 `layout_height` 这几个属性比较简单，我就不解释了。`android:theme` 属性指定了一个 ThemeOverlay.AppCompat.Dark.ActionBar 的主题，其实对于这部分我们也并不陌生，因为之前在 `activity_main.xml` 中给 Toolbar 指定的也是这个主题，只不过这里要实现更加高级的 Toolbar 效果，因此需要将这个主题的指定提到上一层来。`app:contentScrim` 属性用于指定 CollapsingToolbarLayout 在趋于折叠状态以及折叠之后的背景色，其实 CollapsingToolbarLayout 在折叠之后就是一个普通的 Toolbar，那么背景色肯定应该是 colorPrimary 了，具体的效果我们待会儿就能看到。`app:layout_scrollFlags` 属性我们也是见过的，只不过之前是给 Toolbar 指定的，现在也移到外面来了。其中，scroll 表示 CollapsingToolbarLayout 会随着水果内容详情的滚动一起滚动，`exitUntilCollapsed` 表示当 CollapsingToolbarLayout 随着滚动完成折叠之后就保留在界面上，不再移出屏幕。

接下来，我们在 CollapsingToolbarLayout 中定义标题栏的具体内容，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:contentScrim="?attr/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed">
```

```
            <ImageView
                android:id="@+id/fruit_image_view"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:scaleType="centerCrop"
                app:layout_collapseMode="parallax" />

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:layout_collapseMode="pin" />
        </android.support.design.widget.CollapsingToolbarLayout>

    </android.support.design.widget.AppBarLayout>

</android.support.design.widget.CoordinatorLayout>
```

可以看到，我们在 CollapsingToolbarLayout 中定义了一个 ImageView 和一个 Toolbar，也就意味着，这个高级版的标题栏将是由普通的标题栏加上图片组合而成的。这里定义的大多数属性我们都是见过的，就不再解释了，只有一个 `app:layout_collapseMode` 比较陌生。它用于指定当前控件在 CollapsingToolbarLayout 折叠过程中的折叠模式，其中 Toolbar 指定成 pin，表示在折叠的过程中位置始终保持不变，ImageView 指定成 parallax，表示会在折叠的过程中产生一定的错位偏移，这种模式的视觉效果会非常好。

这样我们就将水果标题栏的界面编写完成了，下面开始编写水果内容详情部分。继续修改 activity_fruit.xml 中的代码，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">
        ...
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

    </android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>
```

水果内容详情的最外层布局使用了一个 NestedScrollView，注意它和 AppBarLayout 是平级的。

我们之前在 9.2.1 小节学过 ScrollView 的用法，它允许使用滚动的方式来查看屏幕以外的数据，而 NestedScrollView 在此基础之上还增加了嵌套响应滚动事件的功能。由于 CoordinatorLayout 本身已经可以响应滚动事件了，因此我们在它的内部就需要使用 NestedScrollView 或 RecyclerView 这样的布局。另外，这里还通过 `app:layout_behavior` 属性指定了一个布局行为，这和之前在 RecyclerView 中的用法是一模一样的。

不管是 ScrollView 还是 NestedScrollView，它们的内部都只允许存在一个直接子布局。因此，如果我们想要在里面放入很多东西的话，通常都会先嵌套一个 LinearLayout，然后再在 LinearLayout 中放入具体的内容就可以了，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </LinearLayout>

    </android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>
```

这里我们嵌套了一个垂直方向的 LinearLayout，并将 `layout_width` 设置为 `match_parent`，将 `layout_height` 设置为 `wrap_content`。

接下来在 LinearLayout 中放入具体的内容，这里我准备使用一个 TextView 来显示水果的内容详情，并将 TextView 放在一个卡片式布局当中，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">
```

```
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <android.support.v7.widget.CardView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginBottom="15dp"
                android:layout_marginLeft="15dp"
                android:layout_marginRight="15dp"
                android:layout_marginTop="35dp"
                app:cardCornerRadius="4dp">

                <TextView
                    android:id="@+id/fruit_content_text"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_margin="10dp" />
            </android.support.v7.widget.CardView>

        </LinearLayout>
    </android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>
```

这段代码也没有什么难理解的地方，都是我们学过的知识。需要注意的是，这里为了让界面更加美观，我在 CardView 和 TextView 上都加了一些边距。其中，CardView 的 marginTop 加了 35dp 的边距，这是为下面要编写的东西留出空间。

好的，这样就把水果标题栏和水果内容详情的界面都编写完了，不过我们还可以在界面上再添加一个悬浮按钮。这个悬浮按钮并不是必需的，根据具体的需求添加就可以了，如果加入的话，我们将免费获得一些额外的动画效果。

为了做出示范，我就准备在 activity_fruit.xml 中加入一个悬浮按钮了。这个界面是一个水果详情展示界面，那么我就加入一个表示评论作用的悬浮按钮吧。首先需要提前准备好一个图标，这里我放置了一张 ic_comment.png 到 drawable-xxhdpi 目录下。然后修改 activity_fruit.xml 中的代码，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">
        ...
    </android.support.design.widget.AppBarLayout>
```

```
    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">
        ...
    </android.support.v4.widget.NestedScrollView>

    <android.support.design.widget.FloatingActionButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:src="@drawable/ic_comment"
        app:layout_anchor="@id/appBar"
        app:layout_anchorGravity="bottom|end" />

</android.support.design.widget.CoordinatorLayout>
```

可以看到，这里加入了一个 FloatingActionButton，它和 AppBarLayout 以及 NestedScrollView 是平级的。FloatingActionButton 中使用 `app:layout_anchor` 属性指定了一个锚点，我们将锚点设置为 AppBarLayout，这样悬浮按钮就会出现在水果标题栏的区域内，接着又使用 `app:layout_anchorGravity` 属性将悬浮按钮定位在标题栏区域的右下角。其他一些属性都比较简单，就不再进行解释了。

好了，现在我们终于将整个 activity_fruit.xml 布局都编写完了，内容虽然比较长，但由于是分段编写的，并且每一步我都进行了详细的说明，相信你应该看得很明白吧。

界面完成了之后，接下来我们开始编写功能逻辑，修改 FruitActivity 中的代码，如下所示：

```
public class FruitActivity extends AppCompatActivity {

    public static final String FRUIT_NAME = "fruit_name";

    public static final String FRUIT_IMAGE_ID = "fruit_image_id";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fruit);
        Intent intent = getIntent();
        String fruitName = intent.getStringExtra(FRUIT_NAME);
        int fruitImageId = intent.getIntExtra(FRUIT_IMAGE_ID, 0);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        CollapsingToolbarLayout collapsingToolbar = (CollapsingToolbarLayout)
            findViewById(R.id.collapsing_toolbar);
        ImageView fruitImageView = (ImageView) findViewById(R.id.fruit_image_view);
        TextView fruitContentText = (TextView) findViewById(R.id.fruit_content_
            text);
        setSupportActionBar(toolbar);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
        }
        collapsingToolbar.setTitle(fruitName);
```

```
        Glide.with(this).load(fruitImageId).into(fruitImageView);
        String fruitContent = generateFruitContent(fruitName);
        fruitContentText.setText(fruitContent);
    }

    private String generateFruitContent(String fruitName) {
        StringBuilder fruitContent = new StringBuilder();
        for (int i = 0; i < 500; i++) {
            fruitContent.append(fruitName);
        }
        return fruitContent.toString();
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                finish();
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

FruitActivity 中的代码并不是很复杂。首先，在 `onCreate()`方法中，我们通过 Intent 获取到传入的水果名和水果图片的资源 id，然后通过 `findViewById()`方法拿到刚才在布局文件中定义的各个控件的实例。接着就是使用了 Toolbar 的标准用法，将它作为 ActionBar 显示，并启用 HomeAsUp 按钮。由于 HomeAsUp 按钮的默认图标就是一个返回箭头，这正是我们所期望的，因此就不用再额外设置别的图标了。

接下来开始填充界面上的内容，调用 CollapsingToolbarLayout 的 `setTitle()`方法将水果名设置成当前界面的标题，然后使用 Glide 加载传入的水果图片，并设置到标题栏的 ImageView 上面。接着需要填充水果的内容详情，由于这只是一个示例程序，并不需要什么真实的数据，所以我使用了一个 `generateFruitContent()`方法将水果名循环拼接 500 次，从而生成了一个比较长的字符串，将它设置到了 TextView 上面。

最后，我们在 `onOptionsItemSelected()`方法中处理了 HomeAsUp 按钮的点击事件，当点击了这个按钮时，就调用 `finish()`方法关闭当前的活动，从而返回上一个活动。

所有工作都完成了吗？其实还差最关键的一步，就是处理 RecyclerView 的点击事件，不然的话我们根本就无法打开 FruitActivity。修改 FruitAdapter 中的代码，如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {

    ...

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        if (mContext == null) {
            mContext = parent.getContext();
        }
```

```
            View view = LayoutInflater.from(mContext).inflate(R.layout.fruit_item,
                parent, false);
            final ViewHolder holder = new ViewHolder(view);
            holder.cardView.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    int position = holder.getAdapterPosition();
                    Fruit fruit = mFruitList.get(position);
                    Intent intent = new Intent(mContext, FruitActivity.class);
                    intent.putExtra(FruitActivity.FRUIT_NAME, fruit.getName());
                    intent.putExtra(FruitActivity.FRUIT_IMAGE_ID, fruit.getImageId());
                    mContext.startActivity(intent);
                }
            });
            return holder;
        }

        ...

    }
```

最关键的一步其实也是最简单的，这里我们给 CardView 注册了一个点击事件监听器，然后在点击事件中获取当前点击项的水果名和水果图片资源 id，把它们传入到 Intent 中，最后调用 `startActivity()`方法启动 FruitActivity。

见证奇迹的时刻到了，现在重新运行一下程序，并点击界面上的任意一个水果，比如我点击了葡萄，效果如图 12.17 所示。

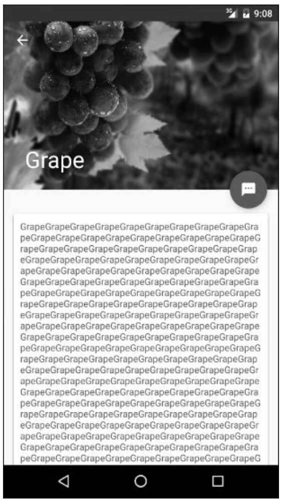


图 12.17 水果的详情展示界面

你没有看错，如此精美的界面就是我们亲手敲出来的。这个界面上的内容分为三部分，水果标题栏、水果内容详情和悬浮按钮，相信你一眼就能将它们区分出来吧。Toolbar 和水果背景图完美地融合到了一起，既保证了图片的展示空间，又不影响 Toolbar 的任何功能，那个向左的箭

头就是用来返回上一个活动的。

不过这并不是全部，真正的好戏还在后头。我们尝试向上拖动水果内容详情，你会发现水果背景图上的标题会慢慢缩小，并且背景图会产生一些错位偏移的效果，如图 12.18 所示。



图 12.18　向上拖动水果内容详情

这是由于用户想要查看水果的内容详情，此时界面的重点在具体的内容上面，因此标题栏就会自动进行折叠，从而节省屏幕空间。

继续向上拖动，直到标题栏变成完全折叠状态，效果如图 12.19 所示。



图 12.19　标题栏变成完全折叠状态

可以看到，标题栏的背景图片不见了，悬浮按钮也自动消失了，现在水果标题栏变成了一个最普通的 Toolbar。这是由于用户正在阅读具体的内容，我们需要给他们提供最充分的阅读空间。而如果这个时候向下拖动水果内容详情，就会执行一个完全相反的动画过程，最终恢复成图 12.17 的界面效果。

不知道你有没有被这个效果所感动呢？在这里，我真心地感谢 Material Design 送给我们的礼物。

12.7.2 充分利用系统状态栏空间

虽说现在水果详情展示界面的效果已经非常华丽了，但这并不代表我们不能再进一步地提升。观察一下图 12.17，你会发现水果的背景图片和系统的状态栏总有一些不搭的感觉，如果我们能将背景图和状态栏融合到一起，那这个视觉体验绝对能提升好几个档次。

只不过很可惜的是，在 Android 5.0 系统之前，我们是无法对状态栏的背景或颜色进行操作的，那个时候也还没有 Material Design 的概念。但是 Android 5.0 及之后的系统都是支持这个功能的，因此这里我们就来实现一个系统差异型的效果，在 Android 5.0 及之后的系统中，使用背景图和状态栏融合的模式，在之前的系统中使用普通的模式。

想要让背景图能够和系统状态栏融合，需要借助 `android:fitsSystemWindows` 这个属性来实现。在 CoordinatorLayout、AppBarLayout、CollapsingToolbarLayout 这种嵌套结构的布局中，将控件的 `android:fitsSystemWindows` 属性指定成 `true`，就表示该控件会出现在系统状态栏里。对应到我们的程序，那就是水果标题栏中的 ImageView 应该设置这个属性了。不过只给 ImageView 设置这个属性是没有用的，我们必须将 ImageView 布局结构中的所有父布局都设置上这个属性才可以，修改 activity_fruit.xml 中的代码，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp"
        android:fitsSystemWindows="true">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            android:fitsSystemWindows="true"
            app:contentScrim="?attr/colorPrimary"
```

```
                app:layout_scrollFlags="scroll|exitUntilCollapsed">

                <ImageView
                    android:id="@+id/fruit_image_view"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:scaleType="centerCrop"
                    android:fitsSystemWindows="true"
                    app:layout_collapseMode="parallax" />

                ...
            </android.support.design.widget.CollapsingToolbarLayout>

        </android.support.design.widget.AppBarLayout>

        ...
    </android.support.design.widget.CoordinatorLayout>
```

但是，即使我们将 `android:fitsSystemWindows` 属性都设置好了还是没有用的，因为还必须在程序的主题中将状态栏颜色指定成透明色才行。指定成透明色的方法很简单，在主题中将 `android:statusBarColor` 属性的值指定成`@android:color/transparent` 就可以了。但问题在于，`android:statusBarColor` 这个属性是从 API 21，也就是 Android 5.0 系统开始才有的，之前的系统无法指定这个属性。那么，系统差异型的功能实现就要从这里开始了。

右击 res 目录→New→Directory，创建一个 values-v21 目录，然后右击 values-v21 目录→New→Values resource file，创建一个 styles.xml 文件。接着对这个文件进行编写，代码如下所示：

```
<resources>

    <style name="FruitActivityTheme" parent="AppTheme">
        <item name="android:statusBarColor">@android:color/transparent</item>
    </style>

</resources>
```

这里我们定义了一个 FruitActivityTheme 主题，它是专门给 FruitActivity 使用的。FruitActivity-Theme 的 parent 主题是 AppTheme，也就是说，它继承了 AppTheme 中的所有特性。然后我们在 FruitActivityTheme 中将状态栏的颜色指定成透明色，由于 values-v21 目录是只有 Android 5.0 及以上的系统才会去读取的，因此这么声明是没有问题的。

但是 Android 5.0 之前的系统却无法识别 FruitActivityTheme 这个主题，因此我们还需要对 values/styles.xml 文件进行修改，如下所示：

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
```

```
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="FruitActivityTheme" parent="AppTheme">
    </style>

</resources>
```

可以看到，这里也定义了一个 FruitActivityTheme 主题，并且 parent 主题也是 AppTheme，但是它的内部是空的。因为 Android 5.0 之前的系统无法指定状态栏的颜色，因此这里什么都不用做就可以了。

最后，我们还需要让 FruitActivity 使用这个主题才可以，修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.materialtest">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <activity
            android:name=".FruitActivity"
            android:theme="@style/FruitActivityTheme">
        </activity>
    </application>

</manifest>
```

这里使用 `android:theme` 属性单独给 FruitActivity 指定了 FruitActivityTheme 这个主题，这样我们就大功告成了。

现在只要是在 Android 5.0 及以上的系统运行 MaterialTest 程序，水果详情展示界面的效果就会如图 12.20 所示。

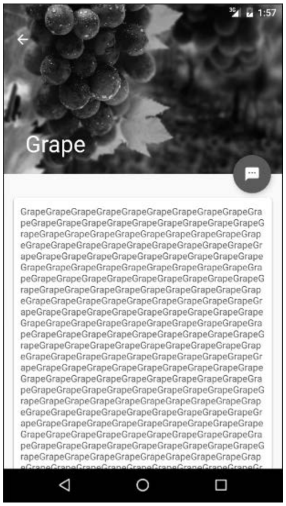


图 12.20　背景图和状态栏融合的效果

相信我，再对比一下图 12.17 的效果，这两种视觉体验绝对不是在一个档次上的。

12.8　小结与点评

学完了本章的所有知识，你有没有觉得无比兴奋呢？反正我是这么觉得的。本章我们的收获实在是太多了，从一个什么都没有的空项目，经过一章的学习，最后实现了一个功能如此丰富、界面如此华丽的应用，还有什么事情比这个更让我们有成就感吗？

本章中我们充分利用了 Design Support 库、support-v4 库、appcompat-v7 库，以及一些开源项目来实现一个了高度 Material 化的应用程序，能将这些库中的相关控件熟练掌握，你的 Material Design 技术就算是合格了。

不过说到底，我仍然还是在以一个开发者的思维给你讲解 Material Design，侧重于如何去实现这些效果。而实际上，Material Design 的设计思维和设计理念才是更加重要的东西，当然这部分内容应该是 UI 设计人员去学习的，如果你也感兴趣的话，可以参考一下 Material Design 的官方文章：https://material.google.com。

现在你已经足足学习了 12 章的内容，对 Android 应用程序开发的理解应该比较深刻了。目前系统性的知识几乎都已经讲完了，但是还有一些零散的高级技巧在等待着你，那么就让我们赶快进入到下一章的学习当中吧。

第 13 章

继续进阶——你还应该掌握的高级技巧

本书的内容虽然已经接近尾声了，但是千万不要因此而放松，现在正是你继续进阶的时机。相信基础性的 Android 知识已经没有太多能够难倒你的了，那么本章中我们就来学习一些你还应该掌握的高级技巧吧。

13.1 全局获取 Context 的技巧

回想这么久以来我们所学的内容，你会发现有很多地方都需要用到 Context，弹出 Toast 的时候需要，启动活动的时候需要，发送广播的时候需要，操作数据库的时候需要，使用通知的时候需要，等等等等。

或许目前你还没有为得不到 Context 而发愁过，因为我们很多的操作都是在活动中进行的，而活动本身就是一个 Context 对象。但是，当应用程序的架构逐渐开始复杂起来的时候，很多的逻辑代码都将脱离 Activity 类，但此时你又恰恰需要使用 Context，也许这个时候你就会感到有些伤脑筋了。

举个例子来说吧，在第 9 章的最佳实践环节，我们编写了一个 HttpUtil 类，在这里将一些通用的网络操作封装了起来，代码如下所示：

```
public class HttpUtil {

    public static void sendHttpRequest(final String address, final
        HttpCallbackListener listener) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                HttpURLConnection connection = null;
                try {
                    URL url = new URL(address);
                    connection = (HttpURLConnection) url.openConnection();
                    connection.setRequestMethod("GET");
```

```
                    connection.setConnectTimeout(8000);
                    connection.setReadTimeout(8000);
                    connection.setDoInput(true);
                    connection.setDoOutput(true);
                    InputStream in = connection.getInputStream();
                    BufferedReader reader = new BufferedReader(new
                        InputStreamReader(in));
                    StringBuilder response = new StringBuilder();
                    String line;
                    while ((line = reader.readLine()) != null) {
                        response.append(line);
                    }
                    if (listener != null) {
                        // 回调onFinish()方法
                        listener.onFinish(response.toString());
                    }
                } catch (Exception e) {
                    if (listener != null) {
                        // 回调onError()方法
                        listener.onError(e);
                    }
                } finally {
                    if (connection != null) {
                        connection.disconnect();
                    }
                }
            }
        }).start();
    }

}
```

这里使用 sendHttpRequest()方法来发送 HTTP 请求显然是没有问题的，并且我们还可以在回调方法中处理服务器返回的数据。但现在我们想对 sendHttpRequest()方法进行一些优化，当检测到网络不存在的时候就给用户一个 Toast 提示，并且不再执行后面的代码。看似一个挺简单的功能，可是却存在一个让人头疼的问题，弹出 Toast 提示需要一个 Context 参数，而我们在 HttpUtil 类中显然是获取不到 Context 对象的，这该怎么办呢？

其实要想快速解决这个问题也很简单，大不了在 sendHttpRequest()方法中添加一个 Context 参数就行了嘛，于是可以将 HttpUtil 中的代码进行如下修改：

```
public class HttpUtil {

    public static void sendHttpRequest(final Context context,
            final String address, final HttpCallbackListener listener) {
        if (!isNetworkAvailable()) {
            Toast.makeText(context, "network is unavailable",
                Toast.LENGTH_SHORT).show();
            return;
        }
        new Thread(new Runnable() {
```

```
                @Override
                public void run() {
                    ...
                }
            }).start();
        }

        private static boolean isNetworkAvailable() {
            ...
        }

    }
```

可以看到,这里在方法中添加了一个 Context 参数,并且假设有一个 isNetworkAvailable()方法用于判断当前网络是否可用，如果网络不可用的话就弹出 Toast 提示，并将方法 return 掉。

虽说这也确实是一种解决方案，但是却有点推卸责任的嫌疑，因为我们将获取 Context 的任务转移给了 sendHttpRequest()方法的调用方，至于调用方能不能得到 Context 对象，那就不是我们需要考虑的问题了。

由此可以看出，在某些情况下，获取 Context 并非是那么容易的一件事，有时候还是挺伤脑筋的。不过别担心，下面我们就来学习一种技巧，让你在项目的任何地方都能够轻松获取到 Context。

Android 提供了一个 Application 类，每当应用程序启动的时候，系统就会自动将这个类进行初始化。而我们可以定制一个自己的 Application 类，以便于管理程序内一些全局的状态信息，比如说全局 Context。

定制一个自己的 Application 其实并不复杂,首先我们需要创建一个 MyApplication 类继承自 Application，代码如下所示：

```
public class MyApplication extends Application {

    private static Context context;

    @Override
    public void onCreate() {
        context = getApplicationContext();
    }

    public static Context getContext() {
        return context;
    }

}
```

可以看到，MyApplication 中的代码非常简单。这里我们重写了父类的 onCreate()方法，并通过调用 getApplicationContext()方法得到了一个应用程序级别的 Context，然后又提供了一个静态的 getContext()方法，在这里将刚才获取到的 Context 进行返回。

接下来我们需要告知系统，当程序启动的时候应该初始化 MyApplication 类，而不是默认的 Application 类。这一步也很简单，在 AndroidManifest.xml 文件的<application>标签下进行指定就可以了，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.networktest"
    android:versionCode="1"
    android:versionName="1.0" >
    ...
    <application
        android:name="com.example.networktest.MyApplication"
        ...>
        ...
    </application>
</manifest>
```

注意这里在指定 MyApplication 的时候一定要加上完整的包名，不然系统将无法找到这个类。

这样我们就已经实现了一种全局获取 Context 的机制，之后不管你想在项目的任何地方使用 Context，只需要调用一下 MyApplication.getContext()就可以了。

那么接下来我们再对 sendHttpRequest()方法进行优化，代码如下所示：

```
public static void sendHttpRequest(final String address, final HttpCallbackListener
    listener) {
    if (!isNetworkAvailable()) {
        Toast.makeText(MyApplication.getContext(), "network is unavailable",
            Toast.LENGTH_SHORT).show();
        return;
    }
    ...
}
```

可以看到，sendHttpRequest()方法不需要再通过传参的方式来得到 Context 对象，而是调用一下 MyApplication.getContext()方法就可以了。有了这个技巧，你再也不用为得不到 Context 对象而发愁了。

然后我们再回顾一下 6.5.2 小节学过的内容，当时为了让 LitePal 可以正常工作，要求必须在 AndroidManifest.xml 中配置如下内容：

```
<application
    android:name="org.litepal.LitePalApplication"
    ...>
    ...
</application>
```

其实道理也是一样的，因为经过这样的配置之后，LitePal 就能在内部自动获取到 Context 了。

不过这里你可能又会产生疑问，如果我们已经配置过了自己的 Application 怎么办？这样岂不是和 LitePalApplication 冲突了？没错，任何一个项目都只能配置一个 Application，

对于这种情况，LitePal 提供了很简单的解决方案，那就是在我们自己的 Application 中去调用 LitePal 的初始化方法就可以了，如下所示：

```
public class MyApplication extends Application {

    private static Context context;

    @Override
    public void onCreate() {
        context = getApplicationContext();
        LitePal.initialize(context);
    }

    public static Context getContext() {
        return context;
    }

}
```

使用这种写法，就相当于我们把全局的 Context 对象通过参数传递给了 LitePal，效果和在 AndroidManifest.xml 中配置 LitePalApplication 是一模一样的。

13.2　使用 Intent 传递对象

Intent 的用法相信你已经比较熟悉了，我们可以借助它来启动活动、发送广播、启动服务等。在进行上述操作的时候，我们还可以在 Intent 中添加一些附加数据，以达到传值的效果，比如在 FirstActivity 中添加如下代码：

```
Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
intent.putExtra("string_data", "hello");
intent.putExtra("int_data", 100);
startActivity(intent);
```

这里调用了 Intent 的 putExtra()方法来添加要传递的数据，之后在 SecondActivity 中就可以得到这些值了，代码如下所示：

```
getIntent().getStringExtra("string_data");
getIntent().getIntExtra("int_data", 0);
```

但是不知道你有没有发现，putExtra()方法中所支持的数据类型是有限的，虽然常用的一些数据类型它都会支持，但是当你想去传递一些自定义对象的时候，就会发现无从下手。不用担心，下面我们就学习一下使用 Intent 来传递对象的技巧。

13.2.1　Serializable 方式

使用 Intent 来传递对象通常有两种实现方式：Serializable 和 Parcelable，本小节中我们先来学习一下第一种实现方式。

Serializable 是序列化的意思，表示将一个对象转换成可存储或可传输的状态。序列化后的对象可以在网络上进行传输，也可以存储到本地。至于序列化的方法也很简单，只需要让一个类去实现 Serializable 这个接口就可以了。

比如说有一个 Person 类，其中包含了 name 和 age 这两个字段，想要将它序列化就可以这样写：

```
public class Person implements Serializable{

    private String name;

    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

}
```

其中，get、set 方法都是用于赋值和读取字段数据的，最重要的部分是在第一行。这里让 Person 类去实现了 Serializable 接口，这样所有的 Person 对象就都是可序列化的了。

接下来在 FirstActivity 中的写法非常简单：

```
Person person = new Person();
person.setName("Tom");
person.setAge(20);
Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
intent.putExtra("person_data", person);
startActivity(intent);
```

可以看到，这里我们创建了一个 Person 的实例，然后就直接将它传入到 putExtra()方法中了。由于 Person 类实现了 Serializable 接口，所以才可以这样写。

接下来在 SecondActivity 中获取这个对象也很简单，写法如下：

```
Person person = (Person) getIntent().getSerializableExtra("person_data");
```

这里调用了 getSerializableExtra()方法来获取通过参数传递过来的序列化对象，接着再将它向下转型成 Person 对象，这样我们就成功实现了使用 Intent 来传递对象的功能了。

13.2.2　Parcelable 方式

除了 Serializable 之外，使用 Parcelable 也可以实现相同的效果，不过不同于将对象进行序列化，Parcelable 方式的实现原理是将一个完整的对象进行分解，而分解后的每一部分都是 Intent 所支持的数据类型，这样也就实现传递对象的功能了。

下面我们来看一下 Parcelable 的实现方式，修改 Person 中的代码，如下所示：

```
public class Person implements Parcelable {

    private String name;

    private int age;

    ...

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(name);  // 写出 name
        dest.writeInt(age);  // 写出 age
    }

    public static final Parcelable.Creator<Person> CREATOR = new Parcelable.
        Creator<Person>() {

        @Override
        public Person createFromParcel(Parcel source) {
            Person person = new Person();
            person.name = source.readString();  // 读取 name
            person.age = source.readInt();  // 读取 age
            return person;
        }

        @Override
        public Person[] newArray(int size) {
            return new Person[size];
        }
    };

}
```

Parcelable 的实现方式要稍微复杂一些。可以看到，首先我们让 Person 类去实现了 Parcelable 接口，这样就必须重写 describeContents()和 writeToParcel()这两个方法。其中 describeContents()方法直接返回 0 就可以了，而 writeToParcel()方法中我们需要调用 Parcel 的 writeXxx()方法，将 Person 类中的字段一一写出。注意，字符串型数据就调用 writeString()方法，整型数据就调用 writeInt()方法，以此类推。

除此之外，我们还必须在 Person 类中提供一个名为 CREATOR 的常量，这里创建了 Parcelable.Creator 接口的一个实现，并将泛型指定为 Person。接着需要重写 createFromParcel()和 newArray()这两个方法，在 createFromParcel()方法中我们要去读取刚才写出的 name 和 age 字段，并创建一个 Person 对象进行返回，其中 name 和 age 都是调用 Parcel 的 readXxx()方法读取到的，注意这里读取的顺序一定要和刚才写出的顺序完全相同。而 newArray()方法中的实现就简单多了，只需要 new 出一个 Person 数组，并使用方法中传入的 size 作为数组大小就可以了。

接下来，在 FirstActivity 中我们仍然可以使用相同的代码来传递 Person 对象，只不过在 SecondActivity 中获取对象的时候需要稍加改动，如下所示：

```
Person person = (Person) getIntent().getParcelableExtra("person_data");
```

注意，这里不再是调用 getSerializableExtra()方法，而是调用 getParcelableExtra()方法来获取传递过来的对象了，其他的地方都完全相同。

这样我们就把使用 Intent 来传递对象的两种实现方式都学习完了，对比一下，Serializable 的方式较为简单，但由于会把整个对象进行序列化，因此效率会比 Parcelable 方式低一些，所以在通常情况下还是更加推荐使用 Parcelable 的方式来实现 Intent 传递对象的功能。

13.3　定制自己的日志工具

早在第 1 章的 1.4 节中我们就已经学过了 Android 日志工具的用法，并且日志工具也确实贯穿了我们整本书的学习，基本上每一章都有用到过。虽然 Android 中自带的日志工具功能非常强大，但也不能说是完全没有缺点，例如在打印日志的控制方面就做得不够好。

打个比方，你正在编写一个比较庞大的项目，期间为了方便调试，在代码的很多地方都打印了大量的日志。最近项目已经基本完成了，但是却有一个非常让人头疼的问题，之前用于调试的那些日志，在项目正式上线之后仍然会照常打印，这样不仅会降低程序的运行效率，还有可能将一些机密性的数据泄露出去。

那该怎么办呢？难道要一行一行地把所有打印日志的代码都删掉？显然这不是什么好点子，不仅费时费力，而且以后你继续维护这个项目的时候可能还会需要这些日志。因此，最理想的情况是能够自由地控制日志的打印，当程序处于开发阶段时就让日志打印出来，当程序上线了之后就把日志屏蔽掉。

看起来好像是挺高级的一个功能，其实并不复杂，我们只需要定制一个自己的日志工具就可以轻松完成了。比如新建一个 LogUtil 类，代码如下所示：

```
public class LogUtil {

    public static final int VERBOSE = 1;

    public static final int DEBUG = 2;
```

```
    public static final int INFO = 3;

    public static final int WARN = 4;

    public static final int ERROR = 5;

    public static final int NOTHING = 6;

    public static int level = VERBOSE;

    public static void v(String tag, String msg) {
        if (level <= VERBOSE) {
            Log.v(tag, msg);
        }
    }

    public static void d(String tag, String msg) {
        if (level <= DEBUG) {
            Log.d(tag, msg);
        }
    }

    public static void i(String tag, String msg) {
        if (level <= INFO) {
            Log.i(tag, msg);
        }
    }

    public static void w(String tag, String msg) {
        if (level <= WARN) {
            Log.w(tag, msg);
        }
    }

    public static void e(String tag, String msg) {
        if (level <= ERROR) {
            Log.e(tag, msg);
        }
    }

}
```

可以看到，我们在 `LogUtil` 中先是定义了 `VERBOSE`、`DEBUG`、`INFO`、`WARN`、`ERROR`、`NOTHING` 这 6 个整型常量，并且它们对应的值都是递增的。然后又定义了一个静态变量 `level`，可以将它的值指定为上面 6 个常量中的任意一个。

接下来我们提供了 `v()`、`d()`、`i()`、`w()`、`e()`这 5 个自定义的日志方法，在其内部分别调用了 `Log.v()`、`Log.d()`、`Log.i()`、`Log.w()`、`Log.e()`这 5 个方法来打印日志，只不过在这些自定义的方法中我们都加入了一个 `if` 判断，只有当 `level` 的值小于或等于对应日志级别值的时候，才会将日志打印出来。

这样就把一个自定义的日志工具创建好了，之后在项目里我们可以像使用普通的日志工具一样使用 LogUtil，比如打印一行 DEBUG 级别的日志就可以这样写：

```
LogUtil.d("TAG", "debug log");
```

打印一行 WARN 级别的日志就可以这样写：

```
LogUtil.w("TAG", "warn log");
```

然后我们只需要修改 level 变量的值，就可以自由地控制日志的打印行为了。比如让 level 等于 VERBOSE 就可以把所有的日志都打印出来，让 level 等于 WARN 就可以只打印警告以上级别的日志，让 level 等于 NOTHING 就可以把所有日志都屏蔽掉。

使用了这种方法之后，刚才所说的那个问题就不复存在了，你只需要在开发阶段将 level 指定成 VERBOSE，当项目正式上线的时候将 level 指定成 NOTHING 就可以了。

13.4　调试 Android 程序

当开发过程中遇到一些奇怪的 bug，但又迟迟定位不出来原因是什么的时候，最好的解决办法就是调试了。调试允许我们逐行地执行代码，并可以实时观察内存中的数据，从而能够比较轻易地查出问题的原因。那么本节中我们就来学习一下使用 Android Studio 来调试 Android 程序的技巧。

还记得在第 5 章的最佳实践环节中编写的那个强制下线程序吗？就让我们通过这个例子来学习一下 Android 程序的调试方法吧。这个程序中有一个登录功能，比如说现在登录出现了问题，我们就可以通过调试来定位问题的原因。

不用多说，调试工作的第一步肯定是添加断点，这里由于我们要调试登录部分的问题，所以断点可以加在登录按钮的点击事件里面。添加断点的方法也很简单，只需要在相应代码行的左边点击一下就可以了，如图 13.1 所示。

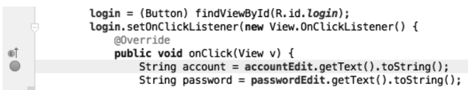


图 13.1　添加断点

如果想要取消这个断点，对着它再次点击就可以了。

添加好了断点，接下来就可以对程序进行调试了，点击 Android Studio 顶部工具栏中的 Debug 按钮（图 13.2 中最右边的按钮），就会使用调试模式来启动程序。



图 13.2　调试按钮

等到程序运行起来的时候，首先会看到一个提示框，如图 13.3 所示。

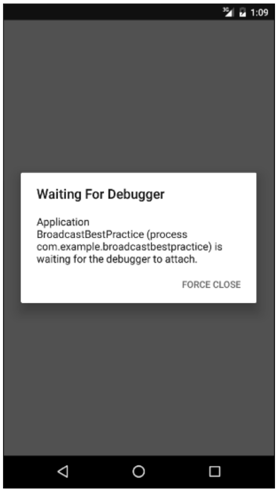


图 13.3　等待调试器提示框

这个框很快就会自动消失，然后在输入框里输入账号和密码，并点击 Login 按钮，这时 Android Studio 就会自动打开 Debug 窗口，如图 13.4 所示。

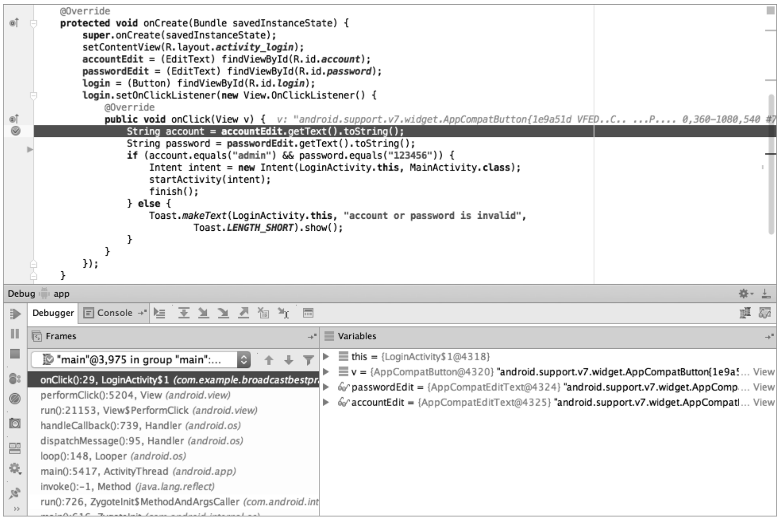


图 13.4　Debug 窗口

接下来每按一次 F8 键，代码就会向下执行一行，并且通过 Variables 视图还可以看到内存中的数据，如图 13.5 所示。



图 13.5　Variables 视图

可以看到，我们从输入框里获取到的账号密码分别是 abc 和 123，而程序里要求正确的账号密码是 admin 和 123456，所以登录才会出现问题。这样我们就通过调试的方式轻松地把问题定位出来了，调试完成之后点击 Debug 窗口中的 Stop 按钮（图 13.6 中最下边的按钮）来结束调试即可。



图 13.6　结束调试按钮

这种调试方式虽然完全可以正常工作，但在调试模式下，程序的运行效率将会大大地降低，如果你的断点加在一个比较靠后的位置，需要执行很多的操作才能运行到这个断点，那么前面这些操作就都会有一些卡顿的感觉。没关系，Android 还提供了另外一种调试的方式，可以让程序随时进入到调试模式，下面我们就来尝试一下。

这次不需要选择调试模式来启动程序了，就使用正常的方式来启动程序。由于现在不是在调试模式下，程序的运行速度比较快，可以先把账号和密码输入好。然后点击 Android Studio 顶部工具栏的 Attach debugger to Android process 按钮（图 13.7 中最左边的按钮）。



图 13.7　动态调试按钮

此时会弹出一个进程选择提示框，如图 13.8 所示。

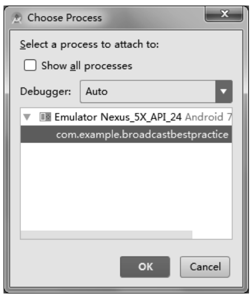


图 13.8　进程选择提示框

这里目前只列出了一个进程，也就是我们当前程序的进程。选中这个进程，然后点击 OK 按钮，就会让这个进程进入到调试模式了。

接下来在程序中点击 Login 按钮，Android Studio 同样也会自动打开 Debug 窗口，之后的流程就都是相同的了。相比起来，第二种调试方式会比第一种更加灵活，也更加常用。

13.5 创建定时任务

Android 中的定时任务一般有两种实现方式，一种是使用 Java API 里提供的 Timer 类，一种是使用 Android 的 Alarm 机制。这两种方式在多数情况下都能实现类似的效果，但 Timer 有一个明显的短板，它并不太适用于那些需要长期在后台运行的定时任务。我们都知道，为了能让电池更加耐用，每种手机都会有自己的休眠策略，Android 手机就会在长时间不操作的情况下自动让 CPU 进入到睡眠状态，这就有可能导致 Timer 中的定时任务无法正常运行。而 Alarm 则具有唤醒 CPU 的功能，它可以保证在大多数情况下需要执行定时任务的时候 CPU 都能正常工作。需要注意，这里唤醒 CPU 和唤醒屏幕完全不是一个概念，千万不要产生混淆。

13.5.1 Alarm 机制

那么首先我们来看一下 Alarm 机制的用法吧，其实并不复杂，主要就是借助了 `AlarmManager` 类来实现的。这个类和 `NotificationManager` 有点类似，都是通过调用 Context 的 `getSystemService()`方法来获取实例的，只是这里需要传入的参数是 `Context.ALARM_ SERVICE`。因此，获取一个 `AlarmManager` 的实例就可以写成：

```
AlarmManager manager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
```

接下来调用 `AlarmManager` 的 `set()`方法就可以设置一个定时任务了，比如说想要设定一个任务在 10 秒钟后执行，就可以写成：

```
long triggerAtTime = SystemClock.elapsedRealtime() + 10 * 1000;
manager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, triggerAtTime, pendingIntent);
```

上面的两行代码你不一定能看得明白，因为 `set()`方法中需要传入的 3 个参数稍微有点复杂，下面我们就来仔细地分析一下。第一个参数是一个整型参数，用于指定 `AlarmManager` 的工作类型，有 4 种值可选，分别是 `ELAPSED_REALTIME`、`ELAPSED_REALTIME_WAKEUP`、`RTC` 和 `RTC_WAKEUP`。其中 `ELAPSED_REALTIME` 表示让定时任务的触发时间从系统开机开始算起，但不会唤醒 CPU。`ELAPSED_REALTIME_WAKEUP` 同样表示让定时任务的触发时间从系统开机开始算起，但会唤醒 CPU。`RTC` 表示让定时任务的触发时间从 1970 年 1 月 1 日 0 点开始算起，但不会唤醒 CPU。`RTC_WAKEUP` 同样表示让定时任务的触发时间从 1970 年 1 月 1 日 0 点开始算起，但会唤醒 CPU。使用 `SystemClock.elapsedRealtime()`方法可以获取到系统开机至今所经历时间的毫秒数，使用 `System.currentTimeMillis()`方法可以获取到 1970 年 1 月 1 日 0 点至今所经历时间的毫秒数。

然后看一下第二个参数，这个参数就好理解多了，就是定时任务触发的时间，以毫秒为单位。如果第一个参数使用的是 ELAPSED_REALTIME 或 ELAPSED_REALTIME_WAKEUP，则这里传入开机至今的时间再加上延迟执行的时间。如果第一个参数使用的是 RTC 或 RTC_WAKEUP，则这里传入 1970 年 1 月 1 日 0 点至今的时间再加上延迟执行的时间。

第三个参数是一个 PendingIntent，对于它你应该已经不会陌生了吧。这里我们一般会调用 getService()方法或者 getBroadcast()方法来获取一个能够执行服务或广播的 PendingIntent。这样当定时任务被触发的时候，服务的 onStartCommand()方法或广播接收器的 onReceive()方法就可以得到执行。

了解了 set()方法的每个参数之后，你应该能想到，设定一个任务在 10 秒钟后执行也可以写成：

```
long triggerAtTime = System.currentTimeMillis() + 10 * 1000;
manager.set(AlarmManager.RTC_WAKEUP, triggerAtTime, pendingIntent);
```

那么，如果我们要实现一个长时间在后台定时运行的服务该怎么做呢？其实很简单，首先新建一个普通的服务，比如把它起名叫 LongRunningService，然后将触发定时任务的代码写到 onStartCommand()方法中，如下所示：

```
public class LongRunningService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                // 在这里执行具体的逻辑操作
            }
        }).start();
        AlarmManager manager = (AlarmManager) getSystemService(ALARM_SERVICE);
        int anHour = 60 * 60 * 1000;  // 这是一小时的毫秒数
        long triggerAtTime = SystemClock.elapsedRealtime() + anHour;
        Intent i = new Intent(this, LongRunningService.class);
        PendingIntent pi = PendingIntent.getService(this, 0, i, 0);
        manager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, triggerAtTime, pi);
        return super.onStartCommand(intent, flags, startId);
    }
}
```

可以看到，我们先是在 onStartCommand()方法中开启了一个子线程，这样就可以在这里执行具体的逻辑操作了。之所以要在子线程里执行逻辑操作，是因为逻辑操作也是需要耗时的，如果放在主线程里执行可能会对定时任务的准确性造成轻微的影响。

创建线程之后的代码就是我们刚刚讲解的 Alarm 机制的用法了，先是获取到了 AlarmManager 的实例，然后定义任务的触发时间为一小时后，再使用 PendingIntent 指定处理定时任务的服务为 LongRunningService，最后调用 set()方法完成设定。

这样我们就将一个长时间在后台定时运行的服务成功实现了。因为一旦启动了 LongRunningService，就会在 onStartCommand()方法里设定一个定时任务，这样一小时后将会再次启动 LongRunningService，从而也就形成了一个永久的循环，保证 LongRunningService 的 onStartCommand()方法可以每隔一小时就执行一次。

最后，只需要在你想要启动定时服务的时候调用如下代码即可：

```
Intent intent = new Intent(context, LongRunningService.class);
context.startService(intent);
```

另外需要注意的是，从 Android 4.4 系统开始，Alarm 任务的触发时间将会变得不准确，有可能会延迟一段时间后任务才能得到执行。这并不是个 bug，而是系统在耗电性方面进行的优化。系统会自动检测目前有多少 Alarm 任务存在，然后将触发时间相近的几个任务放在一起执行，这就可以大幅度地减少 CPU 被唤醒的次数，从而有效延长电池的使用时间。

当然，如果你要求 Alarm 任务的执行时间必须准确无误，Android 仍然提供了解决方案。使用 AlarmManager 的 setExact()方法来替代 set()方法，就基本上可以保证任务能够准时执行了。

13.5.2 Doze 模式

虽然 Android 的每个系统版本都在手机电量方面努力进行优化，不过一直没能解决后台服务泛滥、手机电量消耗过快的问题。于是在 Android 6.0 系统中，谷歌加入了一个全新的 Doze 模式，从而可以极大幅度地延长电池的使用寿命。本小节中我们就来了解一下这个模式，并且掌握一些编程时的注意事项。

首先看一下到底什么是 Doze 模式。当用户的设备是 Android 6.0 或以上系统时，如果该设备未插接电源，处于静止状态（Android 7.0 中删除了这一条件），且屏幕关闭了一段时间之后，就会进入到 Doze 模式。在 Doze 模式下，系统会对 CPU 、网络、Alarm 等活动进行限制，从而延长了电池的使用寿命。

当然，系统并不会一直处于 Doze 模式，而是会间歇性地退出 Doze 模式一小段时间，在这段时间中，应用就可以去完成它们的同步操作、Alarm 任务，等等。图 13.9 完整描述了 Doze 模式的工作过程。

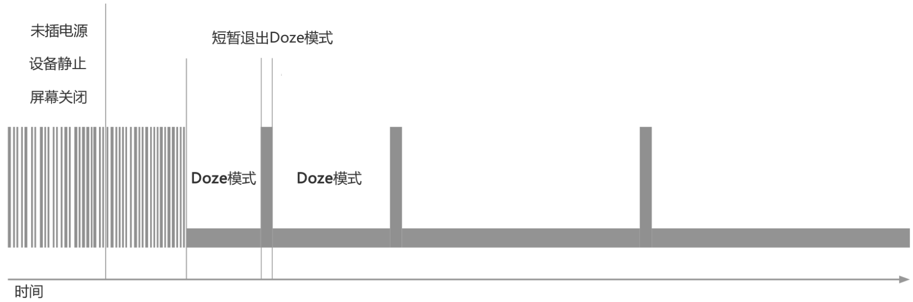


图 13.9　Doze 模式的工作过程

可以看到，随着设备进入 Doze 模式的时间越长，间歇性地退出 Doze 模式的时间间隔也会越长。因为如果设备长时间不使用的话，是没必要频繁退出 Doze 模式来执行同步等操作的，Android 在这些细节上的把控使得电池寿命进一步得到了延长。

接下来我们具体看一看在 Doze 模式下有哪些功能会受到限制吧。

- 网络访问被禁止。
- 系统忽略唤醒 CPU 或者屏幕操作。
- 系统不再执行 WIFI 扫描。
- 系统不再执行同步服务。
- Alarm 任务将会在下次退出 Doze 模式的时候执行。

注意其中的最后一条，也就是说，在 Doze 模式下，我们的 Alarm 任务将会变得不准时。当然，这在大多数情况下都是合理的，因为只有当用户长时间不使用手机的时候才会进入 Doze 模式，通常在这种情况下对 Alarm 任务的准时性要求并没有那么高。

不过，如果你真的有非常特殊的需求，要求 Alarm 任务即使在 Doze 模式下也必须正常执行，Android 还是提供了解决方案。调用 AlarmManager 的 `setAndAllowWhileIdle()`或 `setExactAndAllowWhileIdle()`方法就能让定时任务即使在 Doze 模式下也能正常执行了，这两个方法之间的区别和 `set()`、`setExact()`方法之间的区别是一样的。

13.6　多窗口模式编程

由于手机屏幕大小的限制，传统情况下一个手机只能同时打开一个应用程序，无论是 Android、iOS 还是 Windows Phone 都是如此。我们也早就对此习以为常，认为这是理所当然的事情。而 Android 7.0 系统中却引入了一个非常有特色的功能——多窗口模式，它允许我们在同一个屏幕中同时打开两个应用程序。对于手机屏幕越来越大的今天，这个功能确实是越发重要了，那么本节中我们就将针对这一主题进行学习。

13.6.1 进入多窗口模式

首先你需要知道，我们不用编写任何额外的代码来让应用程序支持多窗口模式。事实上，本书中所编写的所有项目都是支持多窗口模式的。但是这并不意味着我们就不需要对多窗口模式进行学习，因为系统化地了解这些知识点才能编写出在多窗口模式下兼容性更好的程序。

那么先来看一下如何才能进入到多窗口模式。手机的导航栏你肯定是再熟悉不过了，上面一共有 3 个按钮，如图 13.10 所示。



图 13.10　手机导航栏

其中左边的 Back 按钮和中间的 Home 按钮我们都经常使用，但是右边的 Overview 按钮使用得就比较少了。这个按钮的作用是打开一个最近访问过的活动或任务的列表界面，从而能够方便地在多个应用程序之间进行切换，如图 13.11 所示。

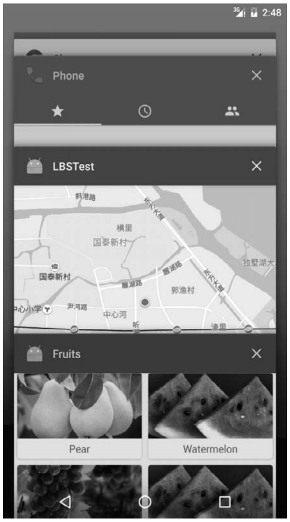


图 13.11　Overview 列表界面

我们可以通过以下两种方式进入多窗口模式。

- 在 Overview 列表界面长按任意一个活动的标题，将该活动拖动到屏幕突出显示的区域，则可以进入多窗口模式。
- 打开任意一个程序，长按 Overview 按钮，也可以进入多窗口模式。

比如说我们首先打开了 MaterialTest 程序，然后长按 Overview 按钮，效果如图 13.12 所示。

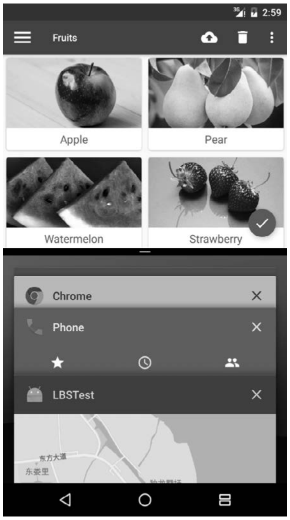


图 13.12　进入多窗口模式

可以看到，现在整个屏幕被分成了上下两个部分，MaterialTest 程序占据了上半屏，下半屏仍然还是一个 Overview 列表界面，另外 Overview 按钮的样式也有了变化。现在我们可以从 Overview 列表中选择任意一个其他程序，比如说这里点击 LBSTest，效果如图 13.13 所示。

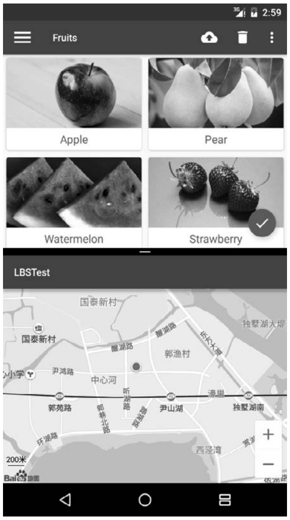


图 13.13　同时打开两个程序

我们还可以将模拟器旋转至水平方向，这样上下分屏的多窗口模式会自动切换成左右分屏的多窗口模式，如图 13.14 所示。

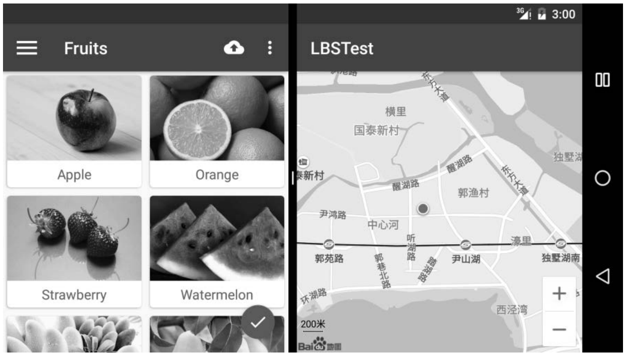


图 13.14 左右分屏的多窗口模式

多窗口模式的用法大概就是这个样子了，我们可以将任意两个应用同时打开，这样就能组合出许多更为丰富的使用场景。比如说刷微博的同时还能时刻关注 QQ 群消息，看电影的同时还能和别人一直聊着微信，等等。如果想要退出多窗口模式，只需要再次长按 Overview 按钮，或者将屏幕中央的分隔线向屏幕任意一个方向拖动到底即可。

可以看出，在多窗口模式下，整个应用的界面会缩小很多，那么编写程序时就应该多考虑使用 match_parent 属性、RecyclerView、ListView、ScrollView 等控件，来让应用的界面能够更好地适配各种不同尺寸的屏幕，尽量不要出现屏幕尺寸变化过大时界面就无法正常显示的情况。

13.6.2 多窗口模式下的生命周期

接下来我们学习一下多窗口模式下的生命周期。其实多窗口模式并不会改变活动原有的生命周期，只是会将用户最近交互过的那个活动设置为运行状态，而将多窗口模式下另外一个可见的活动设置为暂停状态。如果这时用户又去和暂停的活动进行交互，那么该活动就变成运行状态，之前处于运行状态的活动变成暂停状态。

下面我们还是通过一个例子来更加直观地理解多窗口模式下活动的生命周期。首先打开 MaterialTest 项目，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MaterialTest";

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate");
        ...
    }

    @Override
```

```
        protected void onStart() {
            super.onStart();
            Log.d(TAG, "onStart");
        }

        @Override
        protected void onResume() {
            super.onResume();
            Log.d(TAG, "onResume");
        }

        @Override
        protected void onPause() {
            super.onPause();
            Log.d(TAG, "onPause");
        }

        @Override
        protected void onStop() {
            super.onStop();
            Log.d(TAG, "onStop");
        }

        @Override
        protected void onDestroy() {
            super.onDestroy();
            Log.d(TAG, "onDestroy");
        }

        @Override
        protected void onRestart() {
            super.onRestart();
            Log.d(TAG, "onRestart");
        }

        ...
    }
```

这里我们在 Activity 的 7 个生命周期回调方法中分别打印了一句日志。

然后点击 Android Studio 导航栏上的 File→Open Recent→LBSTest，重新打开 LBSTest 项目。修改 MainActivity 的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private static final String TAG = "LBSTest";

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate");
        ...
    }
```

```
    ...

    @Override
    protected void onStart() {
        super.onStart();
        Log.d(TAG, "onStart");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.d(TAG, "onResume");
        mapView.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.d(TAG, "onPause");
        mapView.onPause();
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.d(TAG, "onStop");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
        mLocationClient.stop();
        mapView.onDestroy();
        baiduMap.setMyLocationEnabled(false);
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d(TAG, "onRestart");
    }

    ...
}
```

这里同样也是在 Activity 的 7 个生命周期回调方法中分别打印了一句日志。注意这两处日志的 TAG 是不一样的，方便我们进行区分。

现在，先将 MaterialTest 和 LBSTest 这两个项目的最新代码都运行到模拟器上，然后启动 MaterialTest 程序。这时观察 logcat 中的打印日志（注意要将 logcat 的过滤器选择为 No Filters），如图 13.15 所示。

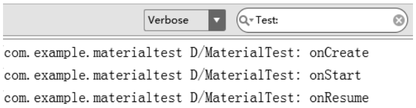


图 13.15　启动 MaterialTest 时的打印日志

可以看到，`onCreate()`、`onStart()`和 `onResume()`方法会依次得到执行，这个也是在我们意料之中的。然后长按 Overview 按钮，进入多窗口模式，此时的打印信息如图 13.16 所示。

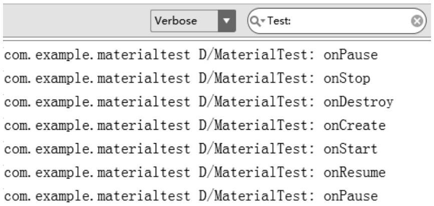


图 13.16　进入多窗口模式时的打印日志

你会发现，MaterialTest 中的 MainActivity 经历了一个重新创建的过程。其实这个是正常现象，因为进入多窗口模式后活动的大小发生了比较大的变化，此时默认是会重新创建活动的。除此之外，像横竖屏切换也是会重新创建活动的。进入多窗口模式后，MaterialTest 变成了暂停状态。

接着在 Overview 列表界面选中 LBSTest 程序，打印信息如图 13.17 所示。

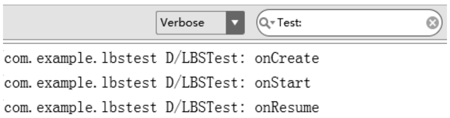


图 13.17　启动 LBSTest 时的打印日志

可以看到，现在 LBSTest 的 `onCreate()`、`onStart()`和 `onResume()`方法依次得到了执行，说明现在 LBSTest 变成了运行状态。

接下来我们可以随意操作一下 MaterialTest 程序，然后观察 logcat 中的打印日志，如图 13.18 所示。



图　13.18

现在 LBSTest 的 `onPause()`方法得到了执行，而 MaterialTest 的 `onResume()`方法得到了执行，说明 LBSTest 变成了暂停状态，MaterialTest 则变成了运行状态，这和我们在本小节开头所分析的生命周期行为是一致的。

了解了多窗口模式下活动的生命周期规则，那么我们在编写程序的时候，就可以将一些关键性的点考虑进去了。比如说，在多窗口模式下，用户仍然可以看到处于暂停状态的应用，那么像视频播放器之类的应用在此时就应该能继续播放视频才对。因此，我们最好不要在活动的 onPause()方法中去处理视频播放器的暂停逻辑，而是应该在 onStop()方法中去处理，并且在 onStart()方法恢复视频的播放。

另外，针对于进入多窗口模式时活动会被重新创建，如果你想改变这一默认行为，可以在 AndroidManifest.xml 中对活动进行如下配置：

```
<activity
    android:name=".MainActivity"
    android:label="Fruits"
    android:configChanges="orientation|keyboardHidden|screenSize|screenLayout">
    ...
</activity>
```

加入了这行配置之后，不管是进入多窗口模式，还是横竖屏切换，活动都不会被重新创建，而是会将屏幕发生变化的事件通知到 Activity 的 onConfigurationChanged()方法当中。因此，如果你想在屏幕发生变化的时候进行相应的逻辑处理，那么在活动中重写 onConfiguration-Changed()方法即可。

13.6.3 禁用多窗口模式

多窗口模式虽然功能非常强大，但是未必就适用于所有的程序。比如说，手机游戏就非常不适合在多窗口模式下运行，很难想象我们如何一边玩着游戏，一边又操作着其他应用。因此，Android 还是给我们提供了禁用多窗口模式的选项，如果你非常不希望自己的应用能够在多窗口模式下运行，那么就可以将这个功能关闭掉。

禁用多窗口模式的方法非常简单，只需要在 AndroidManifest.xml 的<application>或<activity>标签中加入如下属性即可：

```
android:resizeableActivity=["true" | "false"]
```

其中，true 表示应用支持多窗口模式，false 表示应用不支持多窗口模式，如果不配置这个属性，那么默认值为 true。

现在我们将 MaterialTest 程序设置为不支持多窗口模式，如下所示：

```
<application
    ...
    android:resizeableActivity="false">
    ...
</application>
```

重新运行程序，然后长按 Overview 按钮，结果如图 13.19 所示。



图 13.19　不支持多窗口模式时长按 Overview 按钮

可以看到，现在是无法进入到多窗口模式的，而且屏幕下方还会弹出一个 Toast 提示来告知用户，当前应用不支持多窗口模式。

虽说 `android:resizeableActivity` 这个属性的用法很简单，但是它还存在着一个问题，就是这个属性只有当项目的 targetSdkVersion 指定成 24 或者更高的时候才会有用，否则这个属性是无效的。那么比如说我们将项目的 targetSdkVersion 指定成 23，这个时候尝试进入多窗口模式，结果如图 13.20 所示。

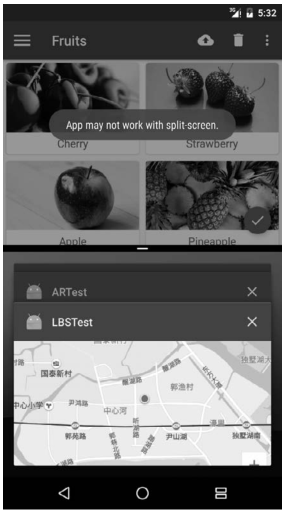


图 13.20　targetSdkVersion 指定成 23 时长按 Overview 按钮

可以看到，虽说界面上弹出了一个提示，告知我们此应用在多窗口模式下可能无法正常工作，但还是进入了多窗口模式。那这样我们就非常头疼了，因为有很多的老项目，它们的 targetSdkVersion 都没有指定到 24，岂不是这些老项目都无法禁用多窗口模式了？

针对这种情况，还有一种解决方案。Android 规定，如果项目指定的 targetSdkVersion 低于 24，并且活动是不允许横竖屏切换的，那么该应用也将不支持多窗口模式。

默认情况下，我们的应用都是可以随着手机的旋转自由地横竖屏切换的，如果想要让应用不允许横竖屏切换，那么就需要在 AndroidManifest.xml 的`<activity>`标签中加入如下配置：

```
android:screenOrientation=["portrait" | "landscape"]
```

其中，`portrait` 表示活动只支持竖屏，`landscape` 表示活动只支持横屏。当然 `android:screenOrientation` 属性中还有很多其他可选值，不过最常用的就是 `portrait` 和 `landscape` 了。

现在我们将 MaterialTest 的 MainActivity 设置为只支持竖屏，如下所示：

```
<activity
    android:name=".MainActivity"
    android:label="Fruits"
    android:screenOrientation="portrait">
    ...
</activity>
```

重新运行程序之后你会发现 MaterialTest 现在不支持横竖屏切换了，此时长按 Overview 按钮会弹出和图 13.19 中一样的提示，说明我们已经成功禁用多窗口模式了。

13.7 Lambda 表达式

Java 8 中着实引入了一些非常有特色的功能，如 Lambda 表达式、stream API、接口默认实现，等等。虽说我们本地安装的 JDK 就是 Java 8 的版本，不过本书中却一直没有使用过任何 Java 8 的新特性。这主要是因为我考虑到你对 Java 8 的新语法规则可能并不熟悉，如果直接应用到项目中的话，容易让代码难以理解，因此这里我就准备单独使用一节的篇幅来对 Java 8 的新特性进行讲解。

虽然刚才已经提到了几个 Java 8 中的新特性，不过现在能够立即应用到项目当中的也就只有 Lambda 表达式而已，因为 stream API 和接口默认实现等特性都只支持 Android 7.0 及以上的系统，我们显然不可能为了使用这些新特性而放弃兼容众多低版本的 Android 手机。而 Lambda 表达式却最低兼容到 Android 2.3 系统，基本上可以算是覆盖所有的 Android 手机了，那么本节中我们就来重点学习一下 Java 8 中的 Lambda 表达式。

Lambda 表达式本质上是一种匿名方法，它既没有方法名，也即没有访问修饰符和返回值类型，使用它来编写代码将会更加简洁，也更加易读。

如果想要在 Android 项目中使用 Lambda 表达式或者 Java 8 的其他新特性，首先我们需要在 app/build.gradle 中添加如下配置：

```
android {
    ...
    defaultConfig {
        ...
        jackOptions.enabled = true
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    ...
}
```

之后就可以开始使用 Lambda 表达式来编写代码了，比如说传统情况下开启一个子线程的写法如下：

```
new Thread(new Runnable() {
    @Override
    public void run() {
        // 处理具体的逻辑
    }
}).start();
```

而使用 Lambda 表达式则可以这样写：

```
new Thread(() -> {
    // 处理具体的逻辑
}).start();
```

是不是很神奇？不管是从代码行数上还是缩进结构上来看，Lambda 表达式的写法明显要更加精简。

那么为什么我们可以使用这么神奇的写法呢？这是因为 `Thread` 类的构造函数接收的参数是一个 `Runnable` 接口，并且该接口中只有一个待实现方法。我们查看一下 `Runnable` 接口的源码，如下所示：

```
public interface Runnable {

    /**
     * Starts executing the active part of the class' code. This method is
     * called when a thread is started that has been created with a class which
     * implements {@code Runnable}.
     */
    public void run();
}
```

凡是这种只有一个待实现方法的接口，都可以使用 Lambda 表达式的写法。比如说，通常创建一个类似于上述接口的匿名类实现需要这样写：

```
Runnable runnable = new Runnable() {
    @Override
    public void run() {
    // 添加具体的实现
```

```
    }
};
```

而有了 Lambda 表达式之后我们就可以这样写了：

```
Runnable runnable1 = () -> {
    // 添加具体的实现
};
```

了解了 Lambda 表达式的基本写法，接下来我们尝试自定义一个接口，然后再使用 Lambda 表达式的方式进行实现。

新建一个 MyListener 接口，代码如下所示：

```
public interface MyListener {

    String doSomething(String a, int b);

}
```

MyListener 接口中也只有一个待实现方法，这和 Runnable 接口的结构是基本一致的。唯一不同的是，MyListener 中的 doSomething()方法是有参数并且有返回值的，那么我们就来看一看这种情况下该如何使用 Lambda 表达式进行实现。

其实写法也是比较相似的,使用 Lambda 表达式创建 MyListener 接口的匿名实现写法如下：

```
MyListener listener = (String a, int b) -> {
    String result = a + b;
    return result;
};
```

可以看到，doSomething()方法的参数直接写在括号里面就可以了，而返回值则仍然像往常一样，写在具体实现的最后一行即可。

另外，Java 还可以根据上下文自动推断出 Lambda 表达式中的参数类型，因此上面的代码也可以简化成如下写法：

```
MyListener listener = (a, b) -> {
    String result = a + b;
    return result;
};
```

Java 将会自动推断出参数 a 是 String 类型，参数 b 是 int 类型，从而使得我们的代码变得更加精简了。

接下来举个具体的例子，比如说现在有一个方法是接收 MyListener 参数的，如下所示：

```
public void hello(MyListener listener) {
    String a = "Hello Lambda";
    int b = 1024;
    String result = listener.doSomething(a, b);
    Log.d("TAG", result);
}
```

我们在调用 `hello()`这个方法的时候就可以这样写：

```
hello((a, b) -> {
    String result = a + b;
    return result;
});
```

那么 `doSomething()`方法就会将 `a` 和 `b` 两个参数进行相加，从而最终的打印结果就会是“`Hello Lambda1024`”。

现在你已经将 Lambda 表达式的写法基本都掌握了，接下来我们看一看在 Android 当中有哪些常用的功能是可以使用 Lambda 表达式进行替换的。

其实只要是符合接口中只有一个待实现方法这个规则的功能，都是可以使用 Lambda 表达式来编写的。除了刚才举例说明的开启子线程之外，还有像设置点击事件之类的功能也是非常适合使用 Lambda 表达式的。

传统情况下，我们给一个按钮设置点击事件需要这样写：

```
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // 处理点击事件
    }
});
```

而使用 Lambda 表达式之后，就可以将代码简化成这个样子了：

```
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener((v) -> {
    // 处理点击事件
});
```

另外，当接口的待实现方法有且只有一个参数的时候，我们还可以进一步简化，将参数外面的括号去掉，如下所示：

```
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(v -> {
    // 处理点击事件
});
```

这样我们就将 Lambda 表达式的主要内容都掌握了。当然，有些人可能并不喜欢 Lambda 表达式这种极简主义的写法。不管你喜欢与否，Java 8 对于哪一种写法都是完全支持的，至于到底要不要使用 Lambda 表达式其实全凭个人，多一种选择总归不是一件坏事情。

13.8　总结

整整 13 章的内容你已经全部学完了！本书的所有知识点也到此结束，是不是感觉有些激动呢？下面就让我们来回顾和总结一下这么久以来学过的所有东西吧。

这 13 章的内容不算很多，但却已经把 Android 中绝大部分比较重要的知识点都覆盖到了。我们从搭建开发环境开始学起，后面逐步学习了四大组件、UI、碎片、数据存储、多媒体、网络、定位服务、Material Design 等内容，本章中又学习了如全局获取 Context、定制日志工具、调试程序、多窗口模式编程、Lambda 表达式等高级技巧，相信你已经从一名初学者蜕变成一位 Android 开发好手了。

不过，虽然你已经储备了足够多的知识，并掌握了很多的最佳实践技巧，但是你还从来没有真正开发过一个完整的项目，也许在将所有学到的知识混合到一起使用的时候，你会感到有些手足无措。因此，前进的脚步仍然不能停下，下一章中我们会结合前面章节所学的内容，一起开发一个天气预报程序。锻炼的机会可千万不能错过，赶快进入到下一章吧。

第 14 章

进入实战——开发酷欧天气

我们将要在本章中编写一个功能较为完整的天气预报程序，学习了这么久的 Android 开发，现在终于到了考核验收的时候了。那么第一步我们需要给这个软件起个好听的名字，这里就叫它酷欧天气吧，英文名就叫作 Cool Weather。确定了名字之后，下面就可以开始动手了。

14.1 功能需求及技术可行性分析

在开始编码之前，我们需要先对程序进行需求分析，想一想酷欧天气中应该具备哪些功能。将这些功能全部整理出来之后，我们才好动手去一一实现。这里我认为酷欧天气中至少应该具备以下功能：

- 可以罗列出全国所有的省、市、县；
- 可以查看全国任意城市的天气信息；
- 可以自由地切换城市，去查看其他城市的天气；
- 提供手动更新以及后台自动更新天气的功能。

虽然看上去只有 4 个主要的功能点，但如果想要全部实现这些功能却需要用到 UI、网络、数据存储、服务等技术，因此还是非常考验你的综合应用能力的。不过好在这些技术在前面的章节中我们全部都学习过了，只要你学得用心，相信完成这些功能对你来说并不难。

分析完了需求之后，接下来就要进行技术可行性分析了。首先需要考虑的一个问题就是，我们如何才能得到全国省市县的数据信息，以及如何才能获取到每个城市的天气信息。比较遗憾的是，现在网上免费的天气预报接口已经越来越少，很多之前可以使用的接口都慢慢关闭掉了，包括本书第 1 版中使用的中国天气网的接口。因此，这次我也是特意用心去找了一些更加稳定的天气预报服务，比如彩云天气以及和风天气都非常不错。这两个天气预报服务虽说都是收费的，但它们每天都提供了一定次数的免费天气预报请求。其中彩云天气的数据更加实时和专业，可以将天气预报精确到分钟级，每天提供 1000 次免费请求；和风天气的数据相对简单一些，比较适合新手学习，每天提供 3000 次免费请求。那么简单起见，这里我们就使用和风天气来作为天气预报的数据来源，每天 3000 次的免费请求对于学习而言已经是相当充足了。

解决了天气数据的问题，接下来还需要解决全国省市县数据的问题。同样，现在网上也没有一个稳定的接口可以使用，那么为了方便你的学习，我专门架设了一台服务器用于提供全国所有省市县的数据信息，从而帮你把道路都铺平了。

那么下面我们来看一下这些接口的具体用法。比如要想罗列出中国所有的省份，只需访问如下地址：

```
http://guolin.tech/api/china
```

服务器会返回我们一段 JSON 格式的数据，其中包含了中国所有的省份名称以及省份 id，如下所示：

```
[{"id":1,"name":"北京"},{"id":2,"name":"上海"},{"id":3,"name":"天津"},
{"id":4,"name":"重庆"},{"id":5,"name":"香港"},{"id":6,"name":"澳门"},
{"id":7,"name":"台湾"},{"id":8,"name":"黑龙江"},{"id":9,"name":"吉林"},
{"id":10,"name":"辽宁"},{"id":11,"name":"内蒙古"},{"id":12,"name":"河北"},
{"id":13,"name":"河南"},{"id":14,"name":"山西"},{"id":15,"name":"山东"},
{"id":16,"name":"江苏"},{"id":17,"name":"浙江"},{"id":18,"name":"福建"},
{"id":19,"name":"江西"},{"id":20,"name":"安徽"},{"id":21,"name":"湖北"},
{"id":22,"name":"湖南"},{"id":23,"name":"广东"},{"id":24,"name":"广西"},
{"id":25,"name":"海南"},{"id":26,"name":"贵州"},{"id":27,"name":"云南"},
{"id":28,"name":"四川"},{"id":29,"name":"西藏"},{"id":30,"name":"陕西"},
{"id":31,"name":"宁夏"},{"id":32,"name":"甘肃"},{"id":33,"name":"青海"},
{"id":34,"name":"新疆"}]
```

可以看到，这是一个 JSON 数组，数组中的每一个元素都代表着一个省份。其中，北京的 id 是 1，上海的 id 是 2。那么如何才能知道某个省内有哪些城市呢？其实也很简单，比如江苏的 id 是 16，访问如下地址即可：

```
http://guolin.tech/api/china/16
```

也就是说，只需要将省份 id 添加到 url 地址的最后面就可以了，现在服务器返回的数据如下：

```
[{"id":113,"name":"南京"},{"id":114,"name":"无锡"},{"id":115,"name":"镇江"},
{"id":116,"name":"苏州"},{"id":117,"name":"南通"},{"id":118,"name":"扬州"},
{"id":119,"name":"盐城"},{"id":120,"name":"徐州"},{"id":121,"name":"淮安"},
{"id":122,"name":"连云港"},{"id":123,"name":"常州"},{"id":124,"name":"泰州"},
{"id":125,"name":"宿迁"}]
```

这样我们就得到江苏省内所有城市的信息了，可以看到，现在返回的数据格式和刚才查看省份信息时返回的数据格式是一样的。相信此时你已经可以举一反三了，比如说苏州的 id 是 116，那么想要知道苏州市下又有哪些县和区的时候，只需访问如下地址：

```
http://guolin.tech/api/china/16/116
```

这次服务器返回的数据如下：

```
[{"id":937,"name":"苏州","weather_id":"CN101190401"},
{"id":938,"name":"常熟","weather_id":"CN101190402"},
{"id":939,"name":"张家港","weather_id":"CN101190403"},
{"id":940,"name":"昆山","weather_id":"CN101190404"},
{"id":941,"name":"吴中","weather_id":"CN101190405"},
```

```
{"id":942,"name":"吴江","weather_id":"CN101190407"},
{"id":943,"name":"太仓","weather_id":"CN101190408"}]
```

通过这种方式，我们就能把全国所有的省、市、县都罗列出来了。那么解决了省市县数据的获取，我们又怎样才能查看到具体的天气信息呢？这就必须要用到每个地区对应的天气 id 了。观察上面返回的数据，你会发现每个县或区都会有一个 weather_id，拿着这个 id 再去访问和风天气的接口，就能够获取到该地区具体的天气信息了。

下面我们来看一下和风天气的接口该如何使用。首先你需要注册一个自己的账号，注册地址是 http://guolin.tech/api/weather/register。注册好了之后使用这个账号登录，就能看到自己的 API Key，以及每天剩余的访问次数了，如图 14.1 所示。



图 14.1　API Key 和每天剩余访问次数

有了 API Key，再配合刚才的 weather_id，我们就能获取到任意城市的天气信息了。比如说苏州的 weather_id 是 CN101190401，那么访问如下接口即可查看苏州的天气信息：

```
http://guolin.tech/api/weather?cityid=CN101190401&key=bc0418b57b2d4918819d3974ac12
85d9
```

其中，cityid 部分填入的就是待查看城市的 weather_id，key 部分填入的就是我们申请到的 API Key。这样，服务器就会把苏州详细的天气信息以 JSON 格式返回给我们了。不过，由于返回的数据过于复杂，这里我做了一下精简处理，如下所示：

```
{
    "HeWeather": [
    {
            "status": "ok",
            "basic": {},
            "aqi": {},
            "now": {},
            "suggestion": {},
            "daily_forecast": []
        }
    ]
}
```

返回数据的格式大体上就是这个样子了，其中 `status` 代表请求的状态，`ok` 表示成功。`basic` 中会包含城市的一些基本信息，`aqi` 中会包含当前空气质量的情况，`now` 中会包含当前的天气信息，`suggestion` 中会包含一些天气相关的生活建议，`daily_forecast` 中会包含未来几天的天气信息。访问 http://guolin.tech/api/weather/doc 这个网址可以查看更加详细的文档说明。

数据都能获取到了之后，接下来就是 JSON 解析的工作了，这对于你来说应该很轻松了吧？

确定了技术完全可行之后，接下来就可以开始编码了。不过别着急，我们准备让酷欧天气成

为一个开源软件，并使用 GitHub 来进行代码托管，因此先让我们进入到本书最后一次的 Git 时间。

14.2　Git 时间——将代码托管到 GitHub 上

经过前面几章的学习，相信你已经可以非常熟练地使用 Git 了。本节依然是 Git 时间，这次我们将会把酷欧天气的代码托管到 GitHub 上面。

GitHub 是全球最大的代码托管网站，主要是借助 Git 来进行版本控制的。任何开源软件都可以免费地将代码提交到 GitHub 上，以零成本的代价进行代码托管。GitHub 的官网地址是 https://github.com/。官网的首页如图 14.2 所示。



图 14.2　GitHub 首页

首先你需要有一个 GitHub 账号才能使用 GitHub 的代码托管功能，点击 Sign up for GitHub 按钮进行注册，然后填入用户名、邮箱和密码，如图 14.3 所示。

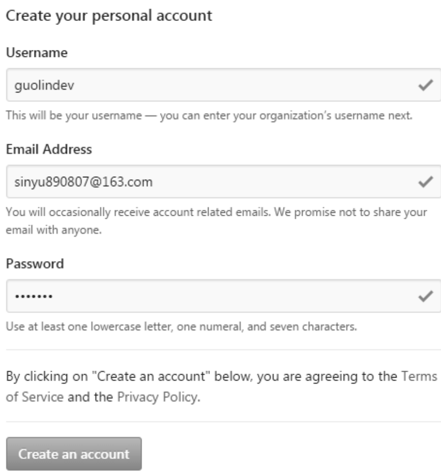


图 14.3　注册账号

点击 Create an account 按钮来创建账户，接下来会让你选择个人计划，收费计划有创建私人版本库的权限，而我们的酷欧天气是开源软件，所以这里选择免费计划就可以了，如图 14.4 所示。

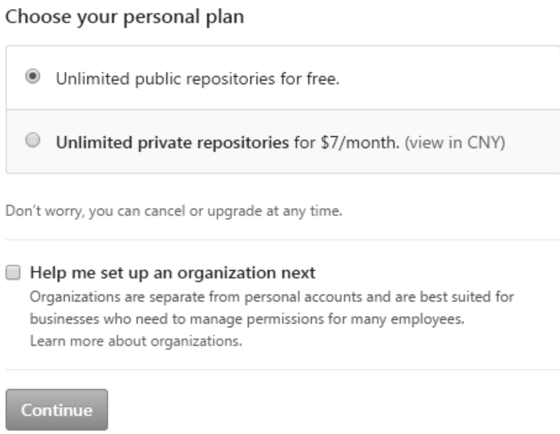


图 14.4　选择免费计划

接着点击 Continue 按钮会进入一个问卷调查界面，如图 14.5 所示。

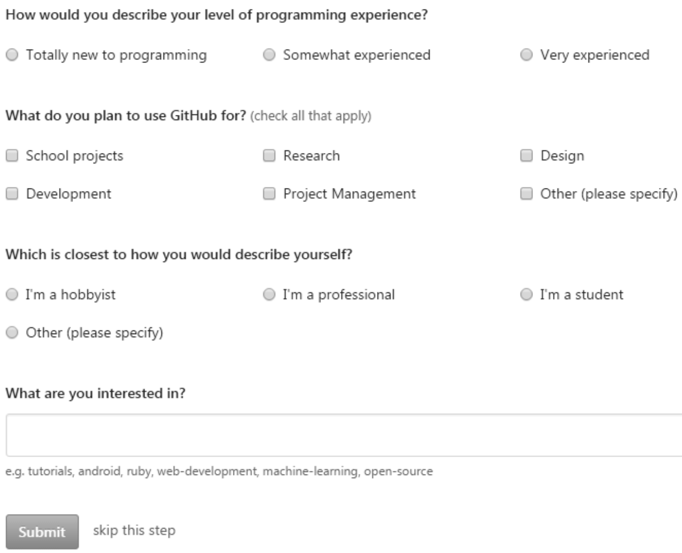


图 14.5　问卷调查界面

如果你对这个有兴趣就填写一下，没兴趣的话直接点击最下方的 skip this step 跳过就可以了。

这样我们就把账号注册好了，会自动跳转到 GitHub 的个人主页，如图 14.6 所示。

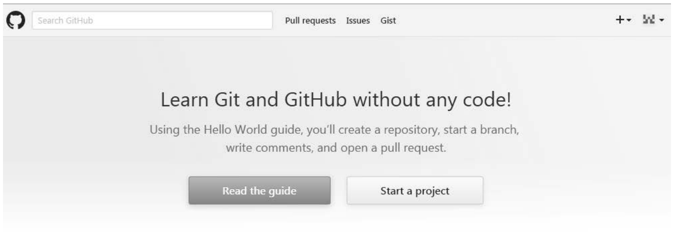


图 14.6　GitHub 个人主页

接下来就可以点击 Start a project 按钮来创建一个版本库了。由于我们是刚刚注册的账号，在创建版本库之前还需要做一下邮箱验证，验证成功之后就能开始创建了。这里将版本库命名为coolweather，然后选择添加一个 Android 项目类型的.gitignore 文件，并使用 Apache License 2.0来作为酷欧天气的开源协议，如图 14.7 所示。

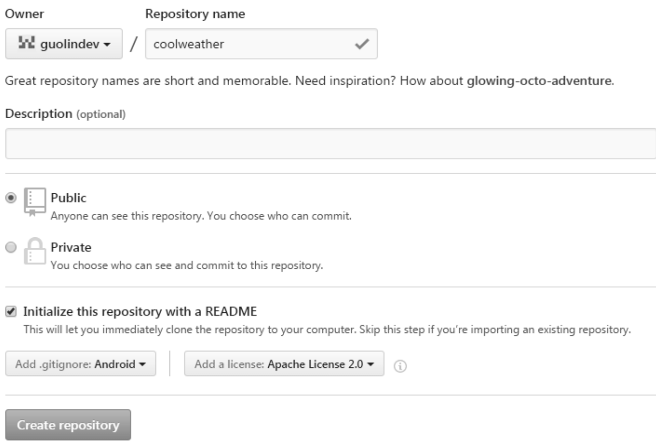


图 14.7　创建版本库

接着点击 Create repository 按钮，coolweather 这个版本库就创建完成了，如图 14.8 所示。版本库主页地址是 https://github.com/guolindev/coolweather。

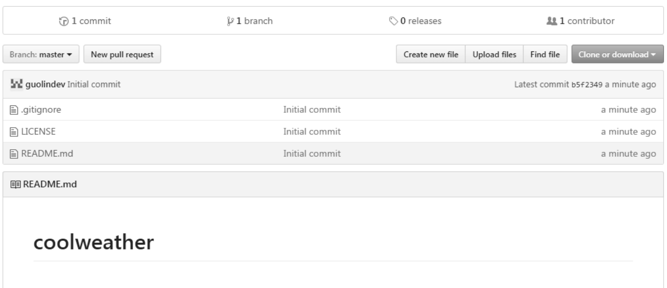


图 14.8　版本库主页

可以看到，GitHub 已经自动帮我们创建了.gitignore、LICENSE 和 README.md 这 3 个文件，其中编辑 README.md 文件中的内容可以修改酷欧天气版本库主页的描述。

创建好了版本库之后，我们就需要创建酷欧天气这个项目了。在 Android Studio 中新建一个 Android 项目，项目名叫作 CoolWeather，包名叫作 com.coolweather.android，如图 14.9 所示。

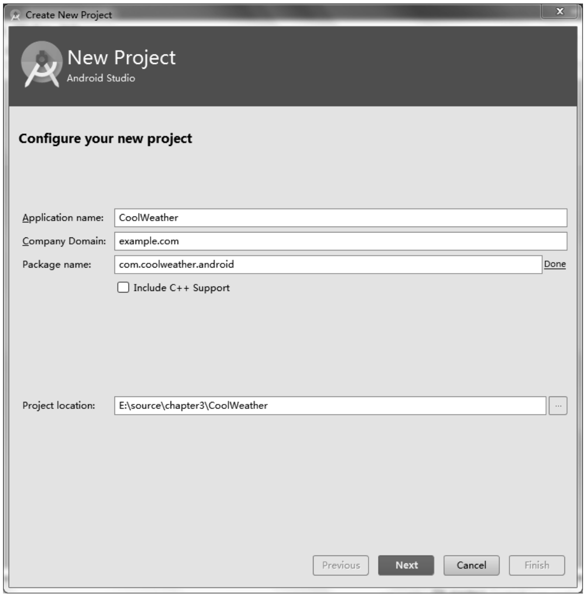


图 14.9　创建 CoolWeather 项目

之后的步骤不用多说，一直点击 Next 就可以完成项目的创建，所有选项都使用默认的就好。

接下来的一步非常重要，我们需要将远程版本库克隆到本地。首先必须知道远程版本库的 Git 地址，点击 Clone or download 按钮就能够看到了，如图 14.10 所示。

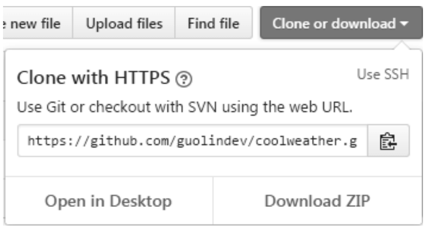
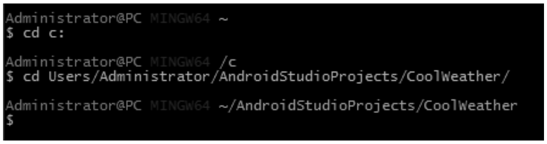


图 14.10　查看版本库的 Git 地址

点击右边的复制按钮可以将版本库的 Git 地址复制到剪贴板，酷欧天气版本库的 Git 地址是 https://github.com/guolindev/coolweather.git。

然后打开 Git Bash 并切换到 CoolWeather 的工程目录下，如图 14.11 所示。



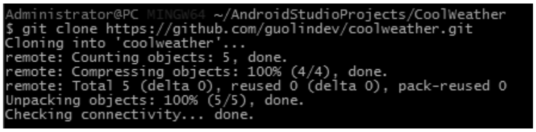
```
Administrator@PC MINGW64 ~
$ cd c:

Administrator@PC MINGW64 /c
$ cd Users/Administrator/AndroidStudioProjects/CoolWeather/

Administrator@PC MINGW64 ~/AndroidStudioProjects/CoolWeather
$
```

图 14.11 在 Git Bash 中进入 CoolWeather 工程目录

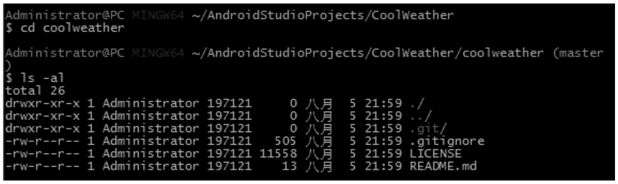
接着输入 git clone https://github.com/guolindev/coolweather.git 来把远程版本库克隆到本地，如图 14.12 所示。



```
Administrator@PC MINGW64 ~/AndroidStudioProjects/CoolWeather
$ git clone https://github.com/guolindev/coolweather.git
Cloning into 'coolweather'...
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
Checking connectivity... done.
```

图 14.12 将远程版本库克隆到本地

看到图中所给的文字提示就表示克隆成功了，并且.gitignore、LICENSE 和 README.md 这 3 个文件也已经被复制到了本地，可以进入到 coolweather 目录，并使用 `ls –al` 命令查看一下，如图 14.13 所示。



```
Administrator@PC MINGW64 ~/AndroidStudioProjects/CoolWeather
$ cd coolweather

Administrator@PC MINGW64 ~/AndroidStudioProjects/CoolWeather/coolweather (master
)
$ ls -al
total 26
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:59 ./
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:59 ../
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:59 .git/
-rw-r--r-- 1 Administrator 197121   505 八月  5 21:59 .gitignore
-rw-r--r-- 1 Administrator 197121 11558 八月  5 21:59 LICENSE
-rw-r--r-- 1 Administrator 197121    13 八月  5 21:59 README.md
```

图 14.13 查看克隆到本地的文件

现在我们需要将这个目录中的所有文件全部复制粘贴到上一层目录中，这样就能将整个 CoolWeather 工程目录添加到版本控制中去了。注意.git 是一个隐藏目录，在复制的时候千万不要漏掉。另外，上一层目录中也有一个.gitignore 文件，我们直接将其覆盖即可。复制完之后可以把 coolweather 目录删除掉，最终 CoolWeather 工程的目录结构如图 14.14 所示。



```
Administrator@PC MINGW64 ~/AndroidStudioProjects/CoolWeather (master)
$ ls -al
total 65
drwxr-xr-x 1 Administrator 197121     0 八月  5 22:04 ./
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:48 ../
drwxr-xr-x 1 Administrator 197121     0 八月  5 22:04 .git/
-rw-r--r-- 1 Administrator 197121   505 八月  5 21:59 .gitignore
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:48 .gradle/
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:59 .idea/
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:49 app/
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:48 build/
-rw-r--r-- 1 Administrator 197121   521 八月  5 21:48 build.gradle
-rw-r--r-- 1 Administrator 197121   942 八月  5 21:48 CoolWeather.iml
drwxr-xr-x 1 Administrator 197121     0 八月  5 21:48 gradle/
-rw-r--r-- 1 Administrator 197121   872 八月  5 21:48 gradle.properties
-rwxr-xr-x 1 Administrator 197121  4971 八月  5 21:48 gradlew*
-rw-r--r-- 1 Administrator 197121  2404 八月  5 21:48 gradlew.bat
-rw-r--r-- 1 Administrator 197121 11558 八月  5 21:59 LICENSE
-rw-r--r-- 1 Administrator 197121   466 八月  5 21:48 local.properties
-rw-r--r-- 1 Administrator 197121    13 八月  5 21:59 README.md
-rw-r--r-- 1 Administrator 197121    16 八月  5 21:48 settings.gradle
```

图 14.14 CoolWeather 工程的目录结构

接下来我们应该把 CoolWeather 项目中现有的文件提交到 GitHub 上面，这就很简单了，先将所有文件添加到版本控制中，如下所示：

```
git add .
```

然后在本地执行提交操作：

```
git commit -m "First commit."
```

最后将提交的内容同步到远程版本库，也就是 GitHub 上面：

```
git push origin master
```

注意，在最后一步的时候 GitHub 要求输入用户名和密码来进行身份校验，这里输入我们注册时填入的用户名和密码就可以了，如图 14.15 所示。

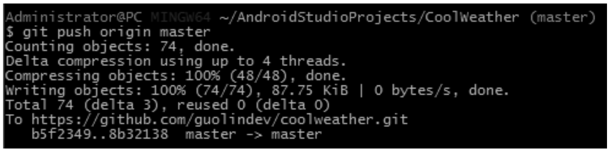


图 14.15　将提交的内容同步到远程版本库

这样就已经同步完成了，现在刷新一下酷欧天气版本库的主页，你会看到刚才提交的那些文件已经存在了，如图 14.16 所示。

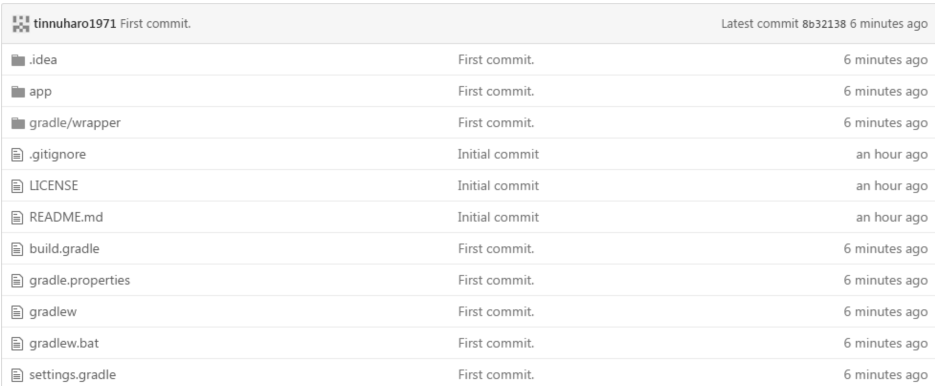


图 14.16　在 GitHub 上查看提交的内容

14.3　创建数据库和表

从本节开始，我们就要真正地动手编码了，为了要让项目能够有更好的结构，这里需要在 com.coolweather.android 包下再新建几个包，如图 14.17 所示。

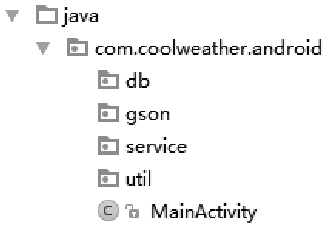


图 14.17　项目的新结构

其中 db 包用于存放数据库模型相关的代码，gson 包用于存放 GSON 模型相关的代码，service 包用于存放服务相关的代码，util 包用于存放工具相关的代码。

根据 14.1 节进行的技术可行性分析，第一阶段我们要做的就是创建好数据库和表，这样从服务器获取到的数据才能够存储到本地。关于数据库和表的创建方式，我们早在第 6 章中就已经学过了。那么为了简化数据库的操作，这里我准备使用 LitePal 来管理酷欧天气的数据库。

首先需要将项目所需的各种依赖库进行声明，编辑 app/build.gradle 文件，在 dependencies 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'org.litepal.android:core:1.4.1'
    compile 'com.squareup.okhttp3:okhttp:3.4.1'
    compile 'com.google.code.gson:gson:2.7'
    compile 'com.github.bumptech.glide:glide:3.7.0'
}
```

这里声明的 4 个库我们之前都是使用过的，LitePal 用于对数据库进行操作，OkHttp 用于进行网络请求，GSON 用于解析 JSON 数据，Glide 用于加载和展示图片。酷欧天气将会对这几个库进行综合运用，这里直接一次性将它们都添加进来。

然后我们来设计一下数据库的表结构，表的设计当然是仁者见仁智者见智，并不是说哪种设计就是最规范最完美的。这里我准备建立 3 张表：province、city、county，分别用于存放省、市、县的数据信息。对应到实体类中的话，就应该建立 `Province`、`City`、`County` 这 3 个类。

那么，在 db 包下新建一个 `Province` 类，代码如下所示：

```
public class Province extends DataSupport {

    private int id;

    private String provinceName;

    private int provinceCode;

    public int getId() {
```

```
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getProvinceName() {
        return provinceName;
    }

    public void setProvinceName(String provinceName) {
        this.provinceName = provinceName;
    }

    public int getProvinceCode() {
        return provinceCode;
    }

    public void setProvinceCode(int provinceCode) {
        this.provinceCode = provinceCode;
}

}
```

其中，id 是每个实体类中都应该有的字段，provinceName 记录省的名字，provinceCode 记录省的代号。另外，LitePal 中的每一个实体类都是必须要继承自 DataSupport 类的。

接着在 db 包下新建一个 City 类，代码如下所示：

```
public class City extends DataSupport {

    private int id;

    private String cityName;

    private int cityCode;

    private int provinceId;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCityName() {
        return cityName;
    }

    public void setCityName(String cityName) {
        this.cityName = cityName;
    }
```

```
    public int getCityCode() {
        return cityCode;
    }

    public void setCityCode(int cityCode) {
        this.cityCode = cityCode;
    }

    public int getProvinceId() {
        return provinceId;
    }

    public void setProvinceId(int provinceId) {
        this.provinceId = provinceId;
    }

}
```

其中，cityName 记录市的名字，cityCode 记录市的代号，provinceId 记录当前市所属省的 id 值。

然后在 db 包下新建一个 County 类，代码如下所示：

```
public class County extends DataSupport {

    private int id;

    private String countyName;

    private String weatherId;

    private int cityId;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCountyName() {
        return countyName;
    }

    public void setCountyName(String countyName) {
        this.countyName = countyName;
    }

    public String getWeatherId() {
        return weatherId;
    }

    public void setWeatherId(String weatherId) {
```

```
        this.weatherId = weatherId;
    }

    public int getCityId() {
        return cityId;
    }

    public void setCityId(int cityId) {
        this.cityId = cityId;
    }

}
```

其中，countyName 记录县的名字，weatherId 记录县所对应的天气 id，cityId 记录当前县所属市的 id 值。

可以看到，实体类的内容都非常简单，就是声明了一些需要的字段，并生成相应的 getter 和 setter 方法就可以了。

接下来需要配置 litepal.xml 文件。右击 app/src/main 目录→New→Directory，创建一个 assets 目录，然后在 assets 目录下再新建一个 litepal.xml 文件，接着编辑 litepal.xml 文件中的内容，如下所示：

```
<litepal>

    <dbname value="cool_weather" />

    <version value="1" />

    <list>
        <mapping class="com.coolweather.android.db.Province" />
        <mapping class="com.coolweather.android.db.City" />
        <mapping class="com.coolweather.android.db.County" />
    </list>

</litepal>
```

这里我们将数据库名指定成 cool_weather，数据库版本指定成 1，并将 Province、City 和 County 这 3 个实体类添加到映射列表当中。

最后还需要再配置一下 LitePalApplication，修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.coolweather.android">

    <application
        android:name="org.litepal.LitePalApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
```

```
        ...
    </application>

</manifest>
```

这样我们就将所有的配置都完成了，数据库和表会在首次执行任意数据库操作的时候自动创建。

好了，第一阶段的代码写到这里就差不多了，我们现在提交一下。首先将所有新增的文件添加到版本控制中：

```
git add .
```

接着执行提交操作：

```
git commit -m "加入创建数据库和表的各项配置。"
```

最后将提交同步到 GitHub 上面：

```
git push origin master
```

OK！第一阶段完工，下面让我们赶快进入到第二阶段的开发工作中吧。

14.4 遍历全国省市县数据

在第二阶段中，我们准备把遍历全国省市县的功能加入，这一阶段需要编写的代码量比较大，你一定要跟上脚步。

我们已经知道，全国所有省市县的数据都是从服务器端获取到的，因此这里和服务器的交互是必不可少的，所以我们可以在 util 包下先增加一个 HttpUtil 类，代码如下所示：

```
public class HttpUtil {

    public static void sendOkHttpRequest(String address, okhttp3.Callback callback) {
        OkHttpClient client = new OkHttpClient();
        Request request = new Request.Builder().url(address).build();
        client.newCall(request).enqueue(callback);
    }

}
```

由于 OkHttp 的出色封装，这里和服务器进行交互的代码非常简单，仅仅 3 行就完成了。现在我们发起一条 HTTP 请求只需要调用 sendOkHttpRequest()方法，传入请求地址，并注册一个回调来处理服务器响应就可以了。

另外，由于服务器返回的省市县数据都是 JSON 格式的，所以我们最好再提供一个工具类来解析和处理这种数据。在 util 包下新建一个 Utility 类，代码如下所示：

```
public class Utility {

    /**
     * 解析和处理服务器返回的省级数据
     */
    public static boolean handleProvinceResponse(String response) {
        if (!TextUtils.isEmpty(response)) {
            try {
                JSONArray allProvinces = new JSONArray(response);
                for (int i = 0; i < allProvinces.length(); i++) {
                    JSONObject provinceObject = allProvinces.getJSONObject(i);
                    Province province = new Province();
                    province.setProvinceName(provinceObject.getString("name"));
                    province.setProvinceCode(provinceObject.getInt("id"));
                    province.save();
                }
                return true;
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return false;
    }

    /**
     * 解析和处理服务器返回的市级数据
     */
    public static boolean handleCityResponse(String response, int provinceId) {
        if (!TextUtils.isEmpty(response)) {
            try {
                JSONArray allCities = new JSONArray(response);
                for (int i = 0; i < allCities.length(); i++) {
                    JSONObject cityObject = allCities.getJSONObject(i);
                    City city = new City();
                    city.setCityName(cityObject.getString("name"));
                    city.setCityCode(cityObject.getInt("id"));
                    city.setProvinceId(provinceId);
                    city.save();
                }
                return true;
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return false;
    }

    /**
     * 解析和处理服务器返回的县级数据
     */
    public static boolean handleCountyResponse(String response, int cityId) {
        if (!TextUtils.isEmpty(response)) {
            try {
                JSONArray allCounties = new JSONArray(response);
```

```
                for (int i = 0; i < allCounties.length(); i++) {
                    JSONObject countyObject = allCounties.getJSONObject(i);
                    County county = new County();
                    county.setCountyName(countyObject.getString("name"));
                    county.setWeatherId(countyObject.getString("weather_id"));
                    county.setCityId(cityId);
                    county.save();
                }
                return true;
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return false;
    }

}
```

可以看到，我们提供了 `handleProvincesResponse()`、`handleCitiesResponse()`、`handleCountiesResponse()`这 3 个方法，分别用于解析和处理服务器返回的省级、市级和县级数据。处理的方式都是类似的，先使用 JSONArray 和 JSONObject 将数据解析出来，然后组装成实体类对象，再调用 `save()`方法将数据存储到数据库当中。由于这里的 JSON 数据结构比较简单，我们就不使用 GSON 来进行解析了。

需要准备的工具类就这么多，现在可以开始写界面了。由于遍历全国省市县的功能我们在后面还会复用，因此就不写在活动里面了，而是写在碎片里面，这样需要复用的时候直接在布局里面引用碎片就可以了。

在 res/layout 目录中新建 choose_area.xml 布局，代码如下所示：

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#fff">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary">

        <TextView
            android:id="@+id/title_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:textColor="#fff"
            android:textSize="20sp"/>

        <Button
```

```
            android:id="@+id/back_button"
            android:layout_width="25dp"
            android:layout_height="25dp"
            android:layout_marginLeft="10dp"
            android:layout_alignParentLeft="true"
            android:layout_centerVertical="true"
            android:background="@drawable/ic_back"/>
    </RelativeLayout>

    <ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

布局文件中的内容并不复杂，我们先是定义了一个头布局来作为标题栏，将布局高度设置为 actionBar 的高度，背景色设置为 colorPrimary。然后在头布局中放置了一个 TextView 用于显示标题内容，放置了一个 Button 用于执行返回操作，注意我已经提前准备好了一张 ic_back.png 图片用于作为按钮的背景图。这里之所以要自己定义标题栏，是因为碎片中最好不要直接使用 ActionBar 或 Toolbar，不然在复用的时候可能会出现一些你不想看到的效果。

接下来在头布局的下面定义了一个 ListView，省市县的数据就将显示在这里。之所以这次使用了 ListView，是因为它会自动给每个子项之间添加一条分隔线，而如果使用 RecyclerView 想实现同样的功能则会比较麻烦，这里我们总是选择最优的实现方案。

接下来也是最关键的一步，我们需要编写用于遍历省市县数据的碎片了。新建 `ChooseAreaFragment` 继承自 Fragment，代码如下所示：

```
public class ChooseAreaFragment extends Fragment {

    public static final int LEVEL_PROVINCE = 0;

    public static final int LEVEL_CITY = 1;

    public static final int LEVEL_COUNTY = 2;

    private ProgressDialog progressDialog;

    private TextView titleText;

    private Button backButton;

    private ListView listView;

    private ArrayAdapter<String> adapter;

    private List<String> dataList = new ArrayList<>();

    /**
     * 省列表
```

```
 */
private List<Province> provinceList;

/**
 * 市列表
 */
private List<City> cityList;

/**
 * 县列表
 */
private List<County> countyList;

/**
 * 选中的省份
 */
private Province selectedProvince;

/**
 * 选中的城市
 */
private City selectedCity;

/**
 * 当前选中的级别
 */
private int currentLevel;


@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.choose_area, container, false);
    titleText = (TextView) view.findViewById(R.id.title_text);
    backButton = (Button) view.findViewById(R.id.back_button);
    listView = (ListView) view.findViewById(R.id.list_view);
    adapter = new ArrayAdapter<>(getContext(), android.R.layout.simple_list_
        item_1, dataList);
    listView.setAdapter(adapter);
    return view;
}

@Override
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
            long id) {
            if (currentLevel == LEVEL_PROVINCE) {
                selectedProvince = provinceList.get(position);
                queryCities();
            } else if (currentLevel == LEVEL_CITY) {
                selectedCity = cityList.get(position);
                queryCounties();
```

```
                }
            }
        });
        backButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (currentLevel == LEVEL_COUNTY) {
                    queryCities();
                } else if (currentLevel == LEVEL_CITY) {
                    queryProvinces();
                }
            }
        });
        queryProvinces();
    }

    /**
     * 查询全国所有的省，优先从数据库查询，如果没有查询到再去服务器上查询
     */
    private void queryProvinces() {
        titleText.setText("中国");
        backButton.setVisibility(View.GONE);
        provinceList = DataSupport.findAll(Province.class);
        if (provinceList.size() > 0) {
            dataList.clear();
            for (Province province : provinceList) {
                dataList.add(province.getProvinceName());
            }
            adapter.notifyDataSetChanged();
            listView.setSelection(0);
            currentLevel = LEVEL_PROVINCE;
        } else {
            String address = "http://guolin.tech/api/china";
            queryFromServer(address, "province");
        }
    }

    /**
     * 查询选中省内所有的市，优先从数据库查询，如果没有查询到再去服务器上查询
     */
    private void queryCities() {
        titleText.setText(selectedProvince.getProvinceName());
        backButton.setVisibility(View.VISIBLE);
        cityList = DataSupport.where("provinceid = ?", String.valueOf(selected
            Province.getId())).find(City.class);
        if (cityList.size() > 0) {
            dataList.clear();
            for (City city : cityList) {
                dataList.add(city.getCityName());
            }
            adapter.notifyDataSetChanged();
            listView.setSelection(0);
            currentLevel = LEVEL_CITY;
        } else {
            int provinceCode = selectedProvince.getProvinceCode();
```

```
            String address = "http://guolin.tech/api/china/" + provinceCode;
            queryFromServer(address, "city");
        }
    }

    /**
     * 查询选中市内所有的县，优先从数据库查询，如果没有查询到再去服务器上查询
     */
    private void queryCounties() {
        titleText.setText(selectedCity.getCityName());
        backButton.setVisibility(View.VISIBLE);
        countyList = DataSupport.where("cityid = ?", String.valueOf(selectedCity.
            getId())).find(County.class);
        if (countyList.size() > 0) {
            dataList.clear();
            for (County county : countyList) {
                dataList.add(county.getCountyName());
            }
            adapter.notifyDataSetChanged();
            listView.setSelection(0);
            currentLevel = LEVEL_COUNTY;
        } else {
            int provinceCode = selectedProvince.getProvinceCode();
            int cityCode = selectedCity.getCityCode();
            String address = "http://guolin.tech/api/china/" + provinceCode + "/" +
                cityCode;
            queryFromServer(address, "county");
        }
    }

    /**
     * 根据传入的地址和类型从服务器上查询省市县数据
     */
    private void queryFromServer(String address, final String type) {
        showProgressDialog();
        HttpUtil.sendOkHttpRequest(address, new Callback() {
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                String responseText = response.body().string();
                boolean result = false;
                if ("province".equals(type)) {
                    result = Utility.handleProvinceResponse(responseText);
                } else if ("city".equals(type)) {
                    result = Utility.handleCityResponse(responseText,
                        selectedProvince.getId());
                } else if ("county".equals(type)) {
                    result = Utility.handleCountyResponse(responseText,
                        selectedCity.getId());
                }
                if (result) {
                    getActivity().runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            closeProgressDialog();
                            if ("province".equals(type)) {
```

```
                            queryProvinces();
                        } else if ("city".equals(type)) {
                            queryCities();
                        } else if ("county".equals(type)) {
                            queryCounties();
                        }
                    }
                });
            }
        }

        @Override
        public void onFailure(Call call, IOException e) {
            // 通过 runOnUiThread()方法回到主线程处理逻辑
            getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    closeProgressDialog();
                    Toast.makeText(getContext(), "加载失败", Toast.LENGTH_SHORT).
                        show();
                }
            });
        }
    });
}

/**
 * 显示进度对话框
 */
private void showProgressDialog() {
    if (progressDialog == null) {
        progressDialog = new ProgressDialog(getActivity());
        progressDialog.setMessage("正在加载...");
        progressDialog.setCanceledOnTouchOutside(false);
    }
    progressDialog.show();
}

/**
 * 关闭进度对话框
 */
private void closeProgressDialog() {
    if (progressDialog != null) {
        progressDialog.dismiss();
    }
}

}
```

这个类里的代码虽然非常多，可是逻辑却不复杂，我们来慢慢理一下。在 `onCreateView()`方法中先是获取到了一些控件的实例，然后去初始化了 ArrayAdapter，并将它设置为 ListView 的适配器。接着在 `onActivityCreated()`方法中给 ListView 和 Button 设置了点击事件，到这里我们的初始化工作就算是完成了。

在 onActivityCreated()方法的最后，调用了 queryProvinces()方法，也就是从这里开始加载省级数据的。queryProvinces()方法中首先会将头布局的标题设置成中国，将返回按钮隐藏起来，因为省级列表已经不能再返回了。然后调用 LitePal 的查询接口来从数据库中读取省级数据，如果读取到了就直接将数据显示到界面上，如果没有读取到就按照 14.1 节讲述的接口组装出一个请求地址，然后调用 queryFromServer()方法来从服务器上查询数据。

queryFromServer()方法中会调用 HttpUtil 的 sendOkHttpRequest()方法来向服务器发送请求，响应的数据会回调到 onResponse()方法中，然后我们在这里去调用 Utility 的 handleProvincesResponse()方法来解析和处理服务器返回的数据，并存储到数据库中。接下来的一步很关键，在解析和处理完数据之后，我们再次调用了 queryProvinces()方法来重新加载省级数据，由于 queryProvinces()方法牵扯到了 UI 操作，因此必须要在主线程中调用，这里借助了 runOnUiThread()方法来实现从子线程切换到主线程。现在数据库中已经存在了数据，因此调用 queryProvinces()就会直接将数据显示到界面上了。

当你点击了某个省的时候会进入到 ListView 的 onItemClick()方法中，这个时候会根据当前的级别来判断是去调用 queryCities()方法还是 queryCounties()方法，queryCities()方法是去查询市级数据，而 queryCounties()方法是去查询县级数据，这两个方法内部的流程和 queryProvinces()方法基本相同，这里就不重复讲解了。

另外还有一点需要注意，在返回按钮的点击事件里，会对当前 ListView 的列表级别进行判断。如果当前是县级列表，那么就返回到市级列表，如果当前是市级列表，那么就返回到省级表列表。当返回到省级列表时，返回按钮会自动隐藏，从而也就不需要再做进一步的处理了。

这样我们就把遍历全国省市县的功能完成了，可是碎片是不能直接显示在界面上的，因此我们还需要把它添加到活动里才行。修改 activity_main.xml 中的代码，如下所示：

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id="@+id/choose_area_fragment"
        android:name="com.coolweather.android.ChooseAreaFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

布局文件很简单，只是定义了一个 FrameLayout，然后将 ChooseAreaFragment 添加进来，并让它充满整个布局。

另外，我们刚才在碎片的布局里面已经自定义了一个标题栏，因此就不再需要原生的 ActionBar 了，修改 res/values/styles.xml 中的代码，如下所示：

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        ...
    </style>

</resources>
```

现在第二阶段的开发工作也完成得差不多了，我们可以运行一下来看看效果。不过在运行之前还有一件事没有做，那就是声明程序所需要的权限。修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.coolweather.android">

    <uses-permission android:name="android.permission.INTERNET" />

    ...

</manifest>
```

由于我们是通过网络接口来获取全国省市县数据的，因此必须要添加访问网络的权限才行。

现在可以运行一下程序了，结果如图 14.18 所示。

可以看到，全国所有省级数据都显示出来了。我们还可以继续查看市级数据，比如点击江苏省，结果如图 14.19 所示。

这个时候标题栏上会出现一个返回按钮，用于返回上一级列表。

然后再点击苏州市查看县级数据，结果如图 14.20 所示。

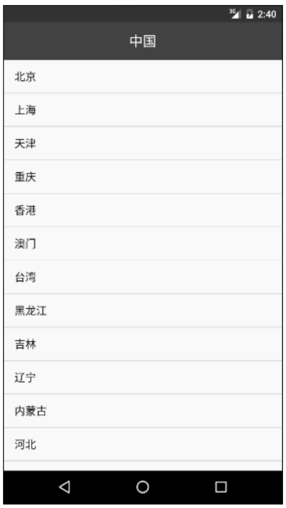


图 14.18　显示省级数据



图 14.19　显示市级数据



图 14.20　显示县级数据

好了，这样第二阶段的开发工作也都完成了，我们仍然要把代码提交一下。

```
git add .
git commit -m "完成遍历省市县三级列表的功能。"
git push origin master
```

到目前为止进度算是相当不错啊，那么我们就趁热打铁，来进行第三阶段的开发工作。

14.5　显示天气信息

在第三阶段中，我们就要开始去查询天气，并且把天气信息显示出来了。由于和风天气返回的 JSON 数据结构非常复杂，如果还使用 JSONObject 来解析就会很麻烦，这里我们就准备借助 GSON 来对天气信息进行解析了。

14.5.1　定义 GSON 实体类

GSON 的用法很简单，解析数据只需要一行代码就能完成了，但前提是要先将数据对应的实体类创建好。由于和风天气返回的数据内容非常多，这里我们不可能将所有的内容都利用起来，因此我筛选了一些比较重要的数据来进行解析。

首先我们回顾一下返回数据的大致格式：

```
{
    "HeWeather": [
        {
            "status": "ok",
            "basic": {},
            "aqi": {},
            "now": {},
            "suggestion": {},
            "daily_forecast": []
        }
    ]
}
```

其中，`basic`、`aqi`、`now`、`suggestion` 和 `daily_forecast` 的内部又都会有具体的内容，那么我们就可以将这 5 个部分定义成 5 个实体类。

下面开始来一个个看，`basic` 中具体内容如下所示：

```
"basic":{
    "city":"苏州",
    "id":"CN101190401",
    "update":{
        "loc":"2016-08-08 21:58"
    }
}
```

其中，`city` 表示城市名，`id` 表示城市对应的天气 id，`update` 中的 `loc` 表示天气的更新时

间。我们按照此结构就可以在 gson 包下建立一个 Basic 类，代码如下所示：

```
public class Basic {

    @SerializedName("city")
    public String cityName;

    @SerializedName("id")
    public String weatherId;

    public Update update;

    public class Update {

        @SerializedName("loc")
        public String updateTime;

    }

}
```

由于 JSON 中的一些字段可能不太适合直接作为 Java 字段来命名，因此这里使用了 @SerializedName 注解的方式来让 JSON 字段和 Java 字段之间建立映射关系。

这样我们就将 Basic 类定义好了，还是挺容易理解的吧？其余的几个实体类也是类似的，我们使用同样的方式来定义就可以了。比如 aqi 中的具体内容如下如示：

```
"aqi":{
    "city":{
        "aqi":"44",
        "pm25":"13"
    }
}
```

那么，在 gson 包下新建一个 AQI 类，代码如下所示：

```
public class AQI {

    public AQICity city;

    public class AQICity {

        public String aqi;

        public String pm25;

    }

}
```

now 中的具体内容如下所示：

```
"now":{
```

```
    "tmp":"29",
    "cond":{
        "txt":"阵雨"
    }
}
```

那么，在 gson 包下新建一个 Now 类，代码如下所示：

```
public class Now {

    @SerializedName("tmp")
    public String temperature;

    @SerializedName("cond")
    public More more;

    public class More {

        @SerializedName("txt")
        public String info;

    }

}
```

suggestion 中的具体内容如下所示：

```
"suggestion":{
    "comf":{
        "txt":"白天天气较热，虽然有雨，但仍然无法削弱较高气温给人们带来的暑意，
            这种天气会让您感到不很舒适。"
    },
    "cw":{
        "txt":"不宜洗车，未来 24 小时内有雨，如果在此期间洗车，雨水和路上的泥水
            可能会再次弄脏您的爱车。"
    },
    "sport":{
        "txt":"有降水，且风力较强，推荐您在室内进行低强度运动；若坚持户外运动，
            请选择避雨防风的地点。"
    }
}
```

那么，在 gson 包下新建一个 Suggestion 类，代码如下所示：

```
public class Suggestion {

    @SerializedName("comf")
    public Comfort comfort;

    @SerializedName("cw")
    public CarWash carWash;

    public Sport sport;
```

```
    public class Comfort {

        @SerializedName("txt")
        public String info;

    }

    public class CarWash {

        @SerializedName("txt")
        public String info;

    }

    public class Sport {

        @SerializedName("txt")
        public String info;

    }

}
```

到目前为止都还比较简单，不过接下来的一项数据就有点特殊了，daily_forecast 中的具体内容如下所示：

```
"daily_forecast":[
    {
        "date":"2016-08-08",
        "cond":{
            "txt_d":"阵雨"
        },
        "tmp":{
            "max":"34",
            "min":"27"
        }
    },
    {
        "date":"2016-08-09",
        "cond":{
            "txt_d":"多云"
        },
        "tmp":{
            "max":"35",
            "min":"29"
        }
    },
    ...
}
```

可以看到，daily_forecast 中包含的是一个数组，数组中的每一项都代表着未来一天的天气信息。针对于这种情况，我们只需要定义出单日天气的实体类就可以了，然后在声明实体类引用的时候使用集合类型来进行声明。

那么在 gson 包下新建一个 Forecast 类，代码如下所示：

```
public class Forecast {

    public String date;

    @SerializedName("tmp")
    public Temperature temperature;

    @SerializedName("cond")
    public More more;

    public class Temperature {

        public String max;

        public String min;

    }

    public class More {

        @SerializedName("txt_d")
        public String info;

    }

}
```

这样我们就把 basic、aqi、now、suggestion 和 daily_forecast 对应的实体类全部都创建好了，接下来还需要再创建一个总的实例类来引用刚刚创建的各个实体类。在 gson 包下新建一个 Weather 类，代码如下所示：

```
public class Weather {

    public String status;

    public Basic basic;

    public AQI aqi;

    public Now now;

    public Suggestion suggestion;

    @SerializedName("daily_forecast")
    public List<Forecast> forecastList;

}
```

在 Weather 类中，我们对 Basic、AQI、Now、Suggestion 和 Forecast 类进行了引用。其中，由于 daily_forecast 中包含的是一个数组，因此这里使用了 List 集合来引用 Forecast 类。

另外，返回的天气数据中还会包含一项 status 数据，成功返回 ok，失败则会返回具体的原因，那么这里也需要添加一个对应的 status 字段。

现在所有的 GSON 实体类都定义好了，接下来我们开始编写天气界面。

14.5.2 编写天气界面

首先创建一个用于显示天气信息的活动。右击 com.coolweather.android 包→New→Activity→Empty Activity，创建一个 WeatherActivity，并将布局名指定成 activity_weather.xml。

由于所有的天气信息都将在同一个界面上显示，因此 activity_weather.xml 会是一个很长的布局文件。那么为了让里面的代码不至于混乱不堪，这里我准备使用 3.4.1 小节学过的引入布局技术，即将界面的不同部分写在不同的布局文件里面，再通过引入布局的方式集成到 activity_weather.xml 中，这样整个布局文件就会显得非常工整。

右击 res/layout→New→Layout resource file，新建一个 title.xml 作为头布局，代码如下所示：

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize">

    <TextView
        android:id="@+id/title_city"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:textColor="#fff"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/title_update_time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="10dp"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"
        android:textColor="#fff"
        android:textSize="16sp"/>

</RelativeLayout>
```

这段代码还是比较简单的，头布局中放置了两个 TextView，一个居中显示城市名，一个居右显示更新时间。

然后新建一个 now.xml 作为当前天气信息的布局，代码如下所示：

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="15dp">

        <TextView
            android:id="@+id/degree_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="end"
            android:textColor="#fff"
            android:textSize="60sp" />

        <TextView
            android:id="@+id/weather_info_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="end"
            android:textColor="#fff"
            android:textSize="20sp" />

    </LinearLayout>
```

当前天气信息的布局中也是放置了两个 TextView，一个用于显示当前气温，一个用于显示天气概况。

然后新建 forecast.xml 作为未来几天天气信息的布局，代码如下所示：

```
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="15dp"
        android:background="#8000">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="15dp"
            android:layout_marginTop="15dp"
            android:text="预报"
            android:textColor="#fff"
            android:textSize="20sp"/>

        <LinearLayout
            android:id="@+id/forecast_layout"
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </LinearLayout>

    </LinearLayout>
```

这里最外层使用 LinearLayout 定义了一个半透明的背景，然后使用 TextView 定义了一个标

题，接着又使用一个 LinearLayout 定义了一个用于显示未来几天天气信息的布局。不过这个布局中并没有放入任何内容，因为这是要根据服务器返回的数据在代码中动态添加的。

为此，我们还需要再定义一个未来天气信息的子项布局，创建 forecast_item.xml 文件，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="15dp">

    <TextView
        android:id="@+id/date_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_weight="2"
        android:textColor="#fff"/>

    <TextView
        android:id="@+id/info_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_weight="1"
        android:gravity="center"
        android:textColor="#fff"/>

    <TextView
        android:id="@+id/max_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:gravity="right"
        android:textColor="#fff"/>

    <TextView
        android:id="@+id/min_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:gravity="right"
        android:textColor="#fff"/>

</LinearLayout>
```

子项布局中放置了 4 个 TextView，一个用于显示天气预报日期，一个用于显示天气概况，另外两个分别用于显示当天的最高温度和最低温度。

然后新建 aqi.xml 作为空气质量信息的布局，代码如下所示：

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="15dp"
    android:background="#8000">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="15dp"
        android:layout_marginTop="15dp"
        android:text="空气质量"
        android:textColor="#fff"
        android:textSize="20sp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="15dp">

        <RelativeLayout
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1">

            <LinearLayout
                android:orientation="vertical"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_centerInParent="true">

                <TextView
                    android:id="@+id/aqi_text"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center"
                    android:textColor="#fff"
                    android:textSize="40sp"
                    />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center"
                    android:text="AQI 指数"
                    android:textColor="#fff"/>

            </LinearLayout>

        </RelativeLayout>

        <RelativeLayout
            android:layout_width="0dp"
            android:layout_height="match_parent"
```

```
            android:layout_weight="1">

            <LinearLayout
                android:orientation="vertical"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_centerInParent="true">

                <TextView
                    android:id="@+id/pm25_text"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center"
                    android:textColor="#fff"
                    android:textSize="40sp"
                    />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center"
                    android:text="PM2.5 指数"
                    android:textColor="#fff"
                    />

            </LinearLayout>

        </RelativeLayout>

    </LinearLayout>

</LinearLayout>
```

这个布局中的代码虽然看上去有点长，但是并不复杂。首先前面都是一样的，使用 LinearLayout 定义了一个半透明的背景，然后使用 TextView 定义了一个标题。接下来，这里使用 LinearLayout 和 RelativeLayout 嵌套的方式实现了一个左右平分并且居中对齐的布局，分别用于显示 AQI 指数和 PM 2.5 指数。相信你只要仔细看一看，这个布局还是很好理解的。

然后新建 suggestion.xml 作为生活建议信息的布局，代码如下所示：

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="15dp"
    android:background="#8000">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="15dp"
        android:layout_marginTop="15dp"
        android:text="生活建议"
```

```
            android:textColor="#fff"
            android:textSize="20sp"/>

        <TextView
            android:id="@+id/comfort_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="15dp"
            android:textColor="#fff" />

        <TextView
            android:id="@+id/car_wash_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="15dp"
            android:textColor="#fff" />

        <TextView
            android:id="@+id/sport_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="15dp"
            android:textColor="#fff" />

    </LinearLayout>
```

这里同样也是先定义了一个半透明的背景和一个标题，然后下面使用了 3 个 TextView 分别用于显示舒适度、洗车指数和运动建议的相关数据。

这样我们就把天气界面上每个部分的布局文件都编写好了，接下来的工作就是将它们引入到 activity_weather.xml 当中，如下所示：

```
    <FrameLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@color/colorPrimary">

        <ScrollView
            android:id="@+id/weather_layout"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scrollbars="none"
            android:overScrollMode="never">

            <LinearLayout
                android:orientation="vertical"
                android:layout_width="match_parent"
                android:layout_height="wrap_content">

                <include layout="@layout/title" />

                <include layout="@layout/now" />
```

```
            <include layout="@layout/forecast" />

            <include layout="@layout/aqi" />

            <include layout="@layout/suggestion" />

        </LinearLayout>

    </ScrollView>

</FrameLayout>
```

可以看到，首先最外层布局使用了一个 FrameLayout，并将它的背景色设置成 colorPrimary。然后在 FrameLayout 中嵌套了一个 ScrollView，这是因为天气界面中的内容比较多，使用 ScrollView 可以允许我们通过滚动的方式查看屏幕以外的内容。

由于 ScrollView 的内部只允许存在一个直接子布局，因此这里又嵌套了一个垂直方向的 LinearLayout，然后在 LinearLayout 中将刚才定义的所有布局逐个引入。

这样我们就将天气界面编写完成了，接下来开始编写业务逻辑，将天气显示到界面上。

14.5.3　将天气显示到界面上

首先需要在 `Utility` 类中添加一个用于解析天气 JSON 数据的方法，如下所示：

```
public class Utility {

    ...

    /**
     * 将返回的 JSON 数据解析成 Weather 实体类
     */
    public static Weather handleWeatherResponse(String response) {
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONArray jsonArray = jsonObject.getJSONArray("HeWeather");
            String weatherContent = jsonArray.getJSONObject(0).toString();
            return new Gson().fromJson(weatherContent, Weather.class);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

可以看到，`handleWeatherResponse()`方法中先是通过 `JSONObject` 和 `JSONArray` 将天气数据中的主体内容解析出来，即如下内容：

```
{
    "status": "ok",
    "basic": {},
    "aqi": {},
```

```
    "now": {},
    "suggestion": {},
    "daily_forecast": []
}
```

然后由于我们之前已经按照上面的数据格式定义过相应的 GSON 实体类，因此只需要通过调用 `fromJson()`方法就能直接将 JSON 数据转换成 `Weather` 对象了。

接下来的工作是我们如何在活动中去请求天气数据，以及将数据展示到界面上。修改 WeatherActivity 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {

    private ScrollView weatherLayout;

    private TextView titleCity;

    private TextView titleUpdateTime;

    private TextView degreeText;

    private TextView weatherInfoText;

    private LinearLayout forecastLayout;

    private TextView aqiText;

    private TextView pm25Text;

    private TextView comfortText;

    private TextView carWashText;

    private TextView sportText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_weather);
        // 初始化各控件
        weatherLayout = (ScrollView) findViewById(R.id.weather_layout);
        titleCity = (TextView) findViewById(R.id.title_city);
        titleUpdateTime = (TextView) findViewById(R.id.title_update_time);
        degreeText = (TextView) findViewById(R.id.degree_text);
        weatherInfoText = (TextView) findViewById(R.id.weather_info_text);
        forecastLayout = (LinearLayout) findViewById(R.id.forecast_layout);
        aqiText = (TextView) findViewById(R.id.aqi_text);
        pm25Text = (TextView) findViewById(R.id.pm25_text);
        comfortText = (TextView) findViewById(R.id.comfort_text);
        carWashText = (TextView) findViewById(R.id.car_wash_text);
        sportText = (TextView) findViewById(R.id.sport_text);
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences
            (this);
        String weatherString = prefs.getString("weather", null);
        if (weatherString != null) {
```

```
            // 有缓存时直接解析天气数据
            Weather weather = Utility.handleWeatherResponse(weatherString);
            showWeatherInfo(weather);
        } else {
            // 无缓存时去服务器查询天气
            String weatherId = getIntent().getStringExtra("weather_id");
            weatherLayout.setVisibility(View.INVISIBLE);
            requestWeather(weatherId);
        }
    }

    /**
     * 根据天气id请求城市天气信息
     */
    public void requestWeather(final String weatherId) {

        String weatherUrl = "http://guolin.tech/api/weather?cityid=" +
            weatherId + "&key=bc0418b57b2d4918819d3974ac1285d9";
        HttpUtil.sendOkHttpRequest(weatherUrl, new Callback() {
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                final String responseText = response.body().string();
                final Weather weather = Utility.handleWeatherResponse(responseText);
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        if (weather != null && "ok".equals(weather.status)) {
                            SharedPreferences.Editor editor = PreferenceManager.
                            getDefaultSharedPreferences(WeatherActivity.this).
                                edit();
                            editor.putString("weather", responseText);
                            editor.apply();
                            showWeatherInfo(weather);
                        } else {
                            Toast.makeText(WeatherActivity.this, "获取天气信息失败",
                            Toast.LENGTH_SHORT).show();
                        }
                    }
                });
            }

            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(WeatherActivity.this, "获取天气信息失败",
                           Toast.LENGTH_SHORT).show();
                    }
                });
            }
        });
    }
```

```
    /**
     * 处理并展示Weather实体类中的数据
     */
    private void showWeatherInfo(Weather weather) {
        String cityName = weather.basic.cityName;
        String updateTime = weather.basic.update.updateTime.split(" ")[1];
        String degree = weather.now.temperature + "℃";
        String weatherInfo = weather.now.more.info;
        titleCity.setText(cityName);
        titleUpdateTime.setText(updateTime);
        degreeText.setText(degree);
        weatherInfoText.setText(weatherInfo);
        forecastLayout.removeAllViews();
        for (Forecast forecast : weather.forecastList) {
            View view = LayoutInflater.from(this).inflate(R.layout.forecast_
                item, forecastLayout, false);
            TextView dateText = (TextView) view.findViewById(R.id.date_text);
            TextView infoText = (TextView) view.findViewById(R.id.info_text);
            TextView maxText = (TextView) view.findViewById(R.id.max_text);
            TextView minText = (TextView) view.findViewById(R.id.min_text);
            dateText.setText(forecast.date);
            infoText.setText(forecast.more.info);
            maxText.setText(forecast.temperature.max);
            minText.setText(forecast.temperature.min);
            forecastLayout.addView(view);
        }
        if (weather.aqi != null) {
            aqiText.setText(weather.aqi.city.aqi);
            pm25Text.setText(weather.aqi.city.pm25);
        }
        String comfort = "舒适度：" + weather.suggestion.comfort.info;
        String carWash = "洗车指数：" + weather.suggestion.carWash.info;
        String sport = "运动建议：" + weather.suggestion.sport.info;
        comfortText.setText(comfort);
        carWashText.setText(carWash);
        sportText.setText(sport);
        weatherLayout.setVisibility(View.VISIBLE);
    }

}
```

这个活动中的代码也比较长，我们还是一步步梳理下。在 `onCreate()`方法中仍然先是去获取一些控件的实例，然后会尝试从本地缓存中读取天气数据。那么第一次肯定是没有缓存的，因此就会从 Intent 中取出天气 id，并调用 `requestWeather()`方法来从服务器请求天气数据。注意，请求数据的时候先将 ScrollView 进行隐藏，不然空数据的界面看上去会很奇怪。

`requestWeather()`方法中先是使用了参数中传入的天气 id 和我们之前申请好的 API Key 拼装出一个接口地址，接着调用 `HttpUtil.sendOkHttpRequest()`方法来向该地址发出请求，服务器会将相应城市的天气信息以 JSON 格式返回。然后我们在 `onResponse()`回调中先调用 `Utility.handleWeatherResponse()`方法将返回的 JSON 数据转换成 `Weather` 对象，再将当前线程切换到主线程。然后进行判断，如果服务器返回的 status 状态是 ok，就说明请求天气成功了，此时将返

回的数据缓存到 SharedPreferences 当中，并调用 showWeatherInfo()方法来进行内容显示。

showWeatherInfo()方法中的逻辑就比较简单了，其实就是从 Weather 对象中获取数据，然后显示到相应的控件上。注意在未来几天天气预报的部分我们使用了一个 for 循环来处理每天的天气信息，在循环中动态加载 forecast_item.xml 布局并设置相应的数据，然后添加到父布局当中。设置完了所有数据之后，记得要将 ScrollView 重新变成可见。

这样我们就将首次进入 WeatherActivity 时的逻辑全部梳理完了，那么当下一次再进入 WeatherActivity 时，由于缓存已经存在了，因此会直接解析并显示天气数据，而不会再次发起网络请求了。

处理完了 WeatherActivity 中的逻辑，接下来我们要做的，就是如何从省市县列表界面跳转到天气界面了，修改 ChooseAreaFragment 中的代码，如下所示：

```
public class ChooseAreaFragment extends Fragment {

    ...

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
                if (currentLevel == LEVEL_PROVINCE) {
                    selectedProvince = provinceList.get(position);
                    queryCities();
                } else if (currentLevel == LEVEL_CITY) {
                    selectedCity = cityList.get(position);
                    queryCounties();
                } else if (currentLevel == LEVEL_COUNTY) {
                    String weatherId = countyList.get(position).getWeatherId();
                    Intent intent = new Intent(getActivity(), WeatherActivity.
                    class);
                    intent.putExtra("weather_id", weatherId);
                    startActivity(intent);
                    getActivity().finish();
                }
            }
        });
        ...
    }

    ...

}
```

非常简单，这里在 onItemClick()方法中加入了一个 if 判断，如果当前级别是 LEVEL_COUNTY，就启动 WeatherActivity，并把当前选中县的天气 id 传递过去。

另外，我们还需要在 MainActivity 中加入一个缓存数据的判断才行。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences
            (this);
        if (prefs.getString("weather", null) != null) {
            Intent intent = new Intent(this, WeatherActivity.class);
            startActivity(intent);
            finish();
        }
    }

}
```

可以看到，这里在 `onCreate()`方法的一开始先从 SharedPreferences 文件中读取缓存数据，如果不为 `null` 就说明之前已经请求过天气数据了，那么就没必要让用户再次选择城市，而是直接跳转到 WeatherActivity 即可。

好了，现在重新运行一下程序，然后选择江苏→苏州→昆山，结果如图 14.21 所示。

然后我们还可以向下滑动查看更多天气信息，如图 14.22 所示。



图 14.21　显示天气信息



图 14.22　查看更多天气信息

14.5.4　获取必应每日一图

虽说现在我们已经把天气界面编写得非常不错了，不过和市场上的一些天气软件的界面相比，仍然还是有一定差距的。出色的天气软件不会像我们现在这样使用一个固定的背景色，而是会根据不同的城市或者天气情况展示不同的背景图片。

当然实现这个功能并不复杂，最重要的是需要有服务器的接口支持。不过我实在是没有精力去准备这样一套完善的服务器接口，那么为了不让我们的天气界面过于单调，这里我准备使用一个巧妙的办法。

必应想必你肯定不会陌生，这是一个由微软开发的搜索引擎网站。这个网站除了提供强大的搜索功能之外，还有一个非常有特色的地方，就是它每天都会在首页展示一张精美的背景图片，如图 14.23 所示。

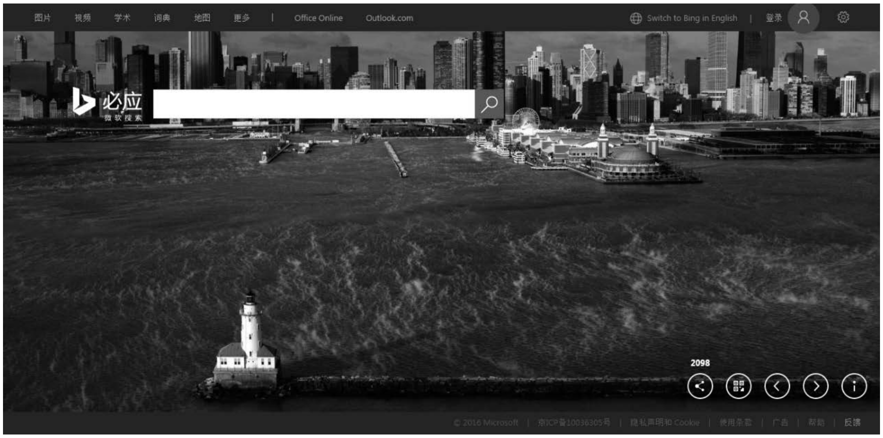


图 14.23　必应的首页

由于这些图片都是由必应精挑细选出来的，并且每天都会变化，如果我们使用它们来作为天气界面的背景图，不仅可以让界面变得更加美观，而且解决了界面一成不变、过于单调的问题。

为此我专门准备了一个获取必应每日一图的接口：http://guolin.tech/api/bing_pic。

访问这个接口，服务器会返回今日的必应背景图链接：

http://cn.bing.com/az/hprichbg/rb/ ChicagoHarborLH_ZH-CN9974330969_1920x1080.jpg。

然后我们再使用 Glide 去加载这张图片就可以了。

总体思路就是这么简单，下面开始来动手实现吧。首先修改 activity_weather.xml 中的代码，如下所示：

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:background="@color/colorPrimary">

    <ImageView
        android:id="@+id/bing_pic_img"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop" />

    <ScrollView
        android:id="@+id/weather_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="none"
        android:overScrollMode="never">

        ...

    </ScrollView>

</FrameLayout>
```

这里我们在 FrameLayout 中添加了一个 ImageView，并且将它的宽和高都设置成 match_parent。由于 FrameLayout 默认情况下会将控件都放置在左上角，因此 ScrollView 会完全覆盖住 ImageView，从而 ImageView 也就成为背景图片了。

接着修改 WeatherActivity 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {

    ...

    private ImageView bingPicImg;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_weather);
        // 初始化各控件
        bingPicImg = (ImageView) findViewById(R.id.bing_pic_img);
        ...
        String bingPic = prefs.getString("bing_pic", null);
        if (bingPic != null) {
            Glide.with(this).load(bingPic).into(bingPicImg);
        } else {
            loadBingPic();
        }
    }

    /**
     * 根据天气 id 请求城市天气信息
     */
    public void requestWeather(final String weatherId) {
```

```
        ...
        loadBingPic();
    }

    /**
     * 加载必应每日一图
     */
    private void loadBingPic() {
        String requestBingPic = "http://guolin.tech/api/bing_pic";
        HttpUtil.sendOkHttpRequest(requestBingPic, new Callback() {
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                final String bingPic = response.body().string();
                SharedPreferences.Editor editor = PreferenceManager.
                    getDefaultSharedPreferences(WeatherActivity.this).edit();
                editor.putString("bing_pic", bingPic);
                editor.apply();
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Glide.with(WeatherActivity.this).load(bingPic).into
                            (bingPicImg);
                    }
                });
            }

            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }
        });
    }

    ...

}
```

可以看到，首先在 onCreate()方法中获取了新增控件 ImageView 的实例，然后尝试从 SharedPreferences 中读取缓存的背景图片。如果有缓存的话就直接使用 Glide 来加载这张图片，如果没有的话就调用 loadBingPic()方法去请求今日的必应背景图。

loadBingPic()方法中的逻辑就非常简单了,先是调用了 HttpUtil.sendOkHttpRequest()方法获取到必应背景图的链接，然后将这个链接缓存到 SharedPreferences 当中，再将当前线程切换到主线程，最后使用 Glide 来加载这张图片就可以了。另外需要注意，在 requestWeather()方法的最后也需要调用一下 loadBingPic()方法，这样在每次请求天气信息的时候同时也会刷新背景图片。现在重新运行一下程序，效果如图 14.24 所示。



图 14.24　在天气界面显示必应背景图

怎么样？虽说只是换了一张背景图而已，但是整个界面的视觉体验就完全不一样了，瞬间提升了好几个档次。而且我们的背景图并不是一成不变的，每天都会是不同的图片，永远给人一种耳目一新的感觉。

不过如果你仔细观察图 14.24，你会发现背景图并没有和状态栏融合到一起，这样的话视觉体验就还是没有达到最佳的效果。虽说我们在 12.7.2 小节已经学习过如何将背景图和状态栏融合到一起，但当时是借助 Design Support 库完成的，而我们这个项目中并没有引入 Design Support 库。

当然如果还是模仿 12.7.2 小节的做法，引入 Design Support 库，然后嵌套 CoordinatorLayout、AppBarLayout、CollapsingToolbarLayout 等布局，也能实现背景图和状态栏融合到一起的效果，不过这样做就过于麻烦了，这里我准备教你另外一种更简单的实现方式。修改 WeatherActivity 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (Build.VERSION.SDK_INT >= 21) {
            View decorView = getWindow().getDecorView();
            decorView.setSystemUiVisibility(
                View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
                | View.SYSTEM_UI_FLAG_LAYOUT_STABLE);
```

```
                getWindow().setStatusBarColor(Color.TRANSPARENT);
            }
            setContentView(R.layout.activity_weather);
            ...
        }

        ...

    }
```

由于这个功能是 Android 5.0 及以上的系统才支持的，因此我们先在代码中做了一个系统版本号的判断，只有当版本号大于或等于 21，也就是 5.0 及以上系统时才会执行后面的代码。

接着我们调用了 `getWindow().getDecorView()`方法拿到当前活动的 DecorView，再调用它的 `setSystemUiVisibility()`方法来改变系统 UI 的显示，这里传入 `View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN` 和 `View.SYSTEM_UI_FLAG_LAYOUT_STABLE` 就表示活动的布局会显示在状态栏上面，最后调用一下 `setStatusBarColor()`方法将状态栏设置成透明色。

仅仅这些代码就可以实现让背景图和状态栏融合到一起的效果了。不过，如果运行一下程序，你会发现还是有些问题，天气界面的头布局几乎和系统状态栏紧贴到一起了，如图 14.25 所示。



图 14.25　头布局和状态栏紧贴在一起

这是由于系统状态栏已经成为我们布局的一部分，因此没有单独为它留出空间。当然，这个问题也是非常好解决的，借助 `android:fitsSystemWindows` 属性就可以了。修改 activity_weather.xml 中的代码，如下所示：

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorPrimary">

    ...

    <ScrollView
        android:id="@+id/weather_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="none"
        android:overScrollMode="never">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fitsSystemWindows="true">

            ...

        </LinearLayout>

    </ScrollView>

</FrameLayout>
```

这里在 ScrollView 的 LinearLayout 中增加了 `android:fitsSystemWindows` 属性，设置成 `true` 就表示会为系统状态栏留出空间。现在重新运行一下代码，效果如图 14.26 所示。



图 14.26　为系统状态栏留出空间

OK，这样第三阶段的开发工作也都完成了，我们把代码提交一下。

```
git add .
git commit -m "加入显示天气信息的功能。"
git push origin master
```

14.6　手动更新天气和切换城市

经过第三阶段的开发，现在酷欧天气的主体功能已经有了，不过你会发现目前存在着一个比较严重的 bug，就是当你选中了某一个城市之后，就没法再去查看其他城市的天气了，即使退出程序，下次进来的时候还会直接跳转到 WeatherActivity。

因此，在第四阶段中我们要加入切换城市的功能，并且为了能够实时获取到最新的天气，我们还会加入手动更新天气的功能。

14.6.1　手动更新天气

先来实现一下手动更新天气的功能。由于我们在上一节中对天气信息进行了缓存，目前每次展示的都是缓存中的数据，因此现在非常需要一种方式能够让用户手动更新天气信息。

至于如何触发更新事件呢？这里我准备采用下拉刷新的方式，正好我们之前也学过下拉刷新的用法，实现起来会比较简单。

首先修改 activity_weather.xml 中的代码，如下所示：

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorPrimary">

    ...

    <android.support.v4.widget.SwipeRefreshLayout
        android:id="@+id/swipe_refresh"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <ScrollView
            android:id="@+id/weather_layout"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scrollbars="none"
            android:overScrollMode="never">

            ...

        </ScrollView>
```

```
    </android.support.v4.widget.SwipeRefreshLayout>

</FrameLayout>
```

可以看到，这里在 ScrollView 的外面又嵌套了一层 SwipeRefreshLayout，这样 ScrollView 就自动拥有下拉刷新功能了。

然后修改 WeatherActivity 中的代码，加入更新天气的处理逻辑，如下所示：

```
public class WeatherActivity extends AppCompatActivity {

    public SwipeRefreshLayout swipeRefresh;
    private String mWeatherId;
    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        swipeRefresh = (SwipeRefreshLayout) findViewById(R.id.swipe_refresh);
        swipeRefresh.setColorSchemeResources(R.color.colorPrimary);
        SharedPreferences prefs = PreferenceManager.
            getDefaultSharedPreferences(this);
        String weatherString = prefs.getString("weather", null);
        if (weatherString != null) {
            // 有缓存时直接解析天气数据
            Weather weather = Utility.handleWeatherResponse(weatherString);
            mWeatherId = weather.basic.weatherId;
            showWeatherInfo(weather);
        } else {
            // 无缓存时去服务器查询天气
            mWeatherId = getIntent().getStringExtra("weather_id");
            weatherLayout.setVisibility(View.INVISIBLE);
            requestWeather(mWeatherId);
        }
        swipeRefresh.setOnRefreshListener(new SwipeRefreshLayout.
            OnRefreshListener() {
            @Override
            public void onRefresh() {
                requestWeather(mWeatherId);
            }
        });
        ...
    }

    /**
     * 根据天气 id 请求城市天气信息
     */
    public void requestWeather(final String weatherId) {

        String weatherUrl = "http://guolin.tech/api/weather?cityid=" +
            weatherId + "&key=bc0418b57b2d4918819d3974ac1285d9";
```

```
        HttpUtil.sendOkHttpRequest(weatherUrl, new Callback() {
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                ...
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        if (weather != null && "ok".equals(weather.status)) {
                            SharedPreferences.Editor editor = PreferenceManager.
                                getDefaultSharedPreferences(WeatherActivity.
                                    this).edit();
                            editor.putString("weather", responseText);
                            editor.apply();
                            mWeatherId = weather.basic.weatherId;
                            showWeatherInfo(weather);
                        } else {
                            Toast.makeText(WeatherActivity.this, "获取天气信息失败",
                               Toast.LENGTH_SHORT).show();
                        }
                        swipeRefresh.setRefreshing(false);
                    }
                });
            }

            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(WeatherActivity.this, "获取天气信息失败",
                            Toast.LENGTH_SHORT).show();
                        swipeRefresh.setRefreshing(false);
                    }
                });
            }
        });
        loadBingPic();
    }

    ...

}
```

修改的代码并不算多，首先在 `onCreate()`方法中获取到了 SwipeRefreshLayout 的实例，然后调用 `setColorSchemeResources()`方法来设置下拉刷新进度条的颜色，这里我们就使用主题中的 colorPrimary 作为进度条的颜色了。接着定义了一个 `mWeatherId` 变量，用于记录城市的天气 id，然后调用 `setOnRefreshListener()`方法来设置一个下拉刷新的监听器，当触发了下拉刷新操作的时候，就会回调这个监听器的 `onRefresh()`方法，我们在这里去调用 `requestWeather()`方法请求天气信息就可以了。

另外不要忘记，当请求结束后，还需要调用 SwipeRefreshLayout 的 `setRefreshing()`方法

并传入 false，用于表示刷新事件结束，并隐藏刷新进度条。

现在重新运行一下程序，并在屏幕的主界面向下拖动，效果如图 14.27 所示。



图 14.27　手动更新天气

更新完天气信息之后，下拉进度条会自动消失。

14.6.2　切换城市

完成了手动更新天气的功能，接下来我们继续实现切换城市功能。

既然是要切换城市，那么就肯定需要遍历全国省市县的数据，而这个功能我们早在 14.4 节就已经完成了，并且当时考虑为了方便后面的复用，特意选择了在碎片当中实现。因此，我们其实只需要在天气界面的布局中引入这个碎片，就可以快速集成切换城市功能了。

虽说实现原理很简单，但是显然我们也不可能让引入的碎片把天气界面遮挡住，这又该怎么办呢？还记得 12.3 节学过的滑动菜单功能吗？将碎片放入到滑动菜单中真是再合适不过了，正常情况下它不占据主界面的任何空间，想要切换城市的时候只需要通过滑动的方式将菜单显示出来就可以了。

下面我们就按照这种思路来实现。首先按照 Material Design 的建议，我们需要在头布局中加入一个切换城市的按钮，不然的话用户可能根本就不知道屏幕的左侧边缘是可以拖动的。修改 title.xml 中的代码，如下所示：

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
        android:layout_height="?attr/actionBarSize">

        <Button
            android:id="@+id/nav_button"
            android:layout_width="30dp"
            android:layout_height="30dp"
            android:layout_marginLeft="10dp"
            android:layout_alignParentLeft="true"
            android:layout_centerVertical="true"
            android:background="@drawable/ic_home" />

        ...

    </RelativeLayout>
```

这里添加了一个 Button 作为切换城市的按钮，并且让它居左显示。另外，我提前准备好了一张图片来作为按钮的背景图。

接着修改 activity_weather.xml 布局来加入滑动菜单功能，如下所示：

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorPrimary">

    ...

    <android.support.v4.widget.DrawerLayout
        android:id="@+id/drawer_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v4.widget.SwipeRefreshLayout
            android:id="@+id/swipe_refresh"
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            ...

        </android.support.v4.widget.SwipeRefreshLayout>

        <fragment
            android:id="@+id/choose_area_fragment"
            android:name="com.coolweather.android.ChooseAreaFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            />

    </android.support.v4.widget.DrawerLayout>

</FrameLayout>
```

可以看到，我们在 SwipeRefreshLayout 的外面又嵌套了一层 DrawerLayout。DrawerLayout 中的第一个子控件用于作为主屏幕中显示的内容，第二个子控件用于作为滑动菜单中显示的内容，因此这里我们在第二个子控件的位置添加了用于遍历省市县数据的碎片。

接下来需要在 WeatherActivity 中加入滑动菜单的逻辑处理，修改 WeatherActivity 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {

    public DrawerLayout drawerLayout;

    private Button navButton;

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        navButton = (Button) findViewById(R.id.nav_button);
        ...
        navButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                drawerLayout.openDrawer(GravityCompat.START);
            }
        });
    }

    ...

}
```

很简单，首先在 `onCreate()`方法中获取到新增的 DrawerLayout 和 Button 的实例，然后在 Button 的点击事件中调用 DrawerLayout 的 `openDrawer()`方法来打开滑动菜单就可以了。

不过现在还没有结束，因为这仅仅是打开了滑动菜单而已，我们还需要处理切换城市后的逻辑才行。这个工作就必须要在 ChooseAreaFragment 中进行了，因为之前选中了某个城市后是跳转到 WeatherActivity 的，而现在由于我们本来就是在 WeatherActivity 当中的，因此并不需要跳转，只是去请求新选择城市的天气信息就可以了。

那么很显然这里我们需要根据 ChooseAreaFragment 的不同状态来进行不同的逻辑处理，修改 ChooseAreaFragment 中的代码，如下所示：

```
public class ChooseAreaFragment extends Fragment {

    ...

    @Override
```

```
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
                if (currentLevel == LEVEL_PROVINCE) {
                    selectedProvince = provinceList.get(position);
                    queryCities();
                } else if (currentLevel == LEVEL_CITY) {
                    selectedCity = cityList.get(position);
                    queryCounties();
                } else if (currentLevel == LEVEL_COUNTY) {
                    String weatherId = countyList.get(position).getWeatherId();
                    if (getActivity() instanceof MainActivity) {
                        Intent intent = new Intent(getActivity(), WeatherActivity.
                            class);
                        intent.putExtra("weather_id", weatherId);
                        startActivity(intent);
                        getActivity().finish();
                    } else if (getActivity() instanceof WeatherActivity) {
                        WeatherActivity activity = (WeatherActivity) getActivity();
                        activity.drawerLayout.closeDrawers();
                        activity.swipeRefresh.setRefreshing(true);
                        activity.requestWeather(weatherId);
                    }
                }
            }
        });
        ...
    }

    ...

}
```

这里我使用了一个 Java 中的小技巧，`instanceof` 关键字可以用来判断一个对象是否属于某个类的实例。我们在碎片中调用 `getActivity()`方法，然后配合 `instanceof` 关键字，就能轻松判断出该碎片是在 MainActivity 当中，还是在 WeatherActivity 当中。如果是在 MainActivity 当中，那么处理逻辑不变。如果是在 WeatherActivity 当中，那么就关闭滑动菜单，显示下拉刷新进度条，然后请求新城市的天气信息。

这样我们就把切换城市的功能全部完成了，现在可以重新运行一下程序，效果如图 14.28 所示。



图 14.28 拥有切换城市按钮的天气界面

可以看到，标题栏上多出了一个用于切换城市的按钮。点击该按钮，或者在屏幕的左侧边缘进行拖动，就能让滑动菜单界面显示出来了，如图 14.29 所示。



图 14.29 显示滑动菜单界面

然后我们就可以在这里切换其他城市了。选中城市之后滑动菜单会自动关闭，并且主界面上的天气信息也会更新成你选择的那个城市。

这样，第四阶段的开发任务也完成了。当然，仍然不要忘记提交代码。

```
git add .
git commit -m "新增切换城市和手动更新天气的功能。"
git push origin master
```

14.7　后台自动更新天气

为了要让酷欧天气更加智能，在第五阶段我们准备加入后台自动更新天气的功能，这样就可以尽可能地保证用户每次打开软件时看到的都是最新的天气信息。

要想实现上述功能，就需要创建一个长期在后台运行的定时任务，这个功能肯定是难不倒你的，因为我们在 13.5 节中就已经学习过了。

首先在 service 包下新建一个服务，右击 com.coolweather.android.service→New→Service→Service，创建一个 AutoUpdateService，并将 Exported 和 Enabled 这两个属性都勾中。然后修改 AutoUpdateService 中的代码，如下所示：

```
public class AutoUpdateService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        updateWeather();
        updateBingPic();
        AlarmManager manager = (AlarmManager) getSystemService(ALARM_SERVICE);
        int anHour = 8 * 60 * 60 * 1000; // 这是 8 小时的毫秒数
        long triggerAtTime = SystemClock.elapsedRealtime() + anHour;
        Intent i = new Intent(this, AutoUpdateService.class);
        PendingIntent pi = PendingIntent.getService(this, 0, i, 0);
        manager.cancel(pi);
        manager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, triggerAtTime, pi);
        return super.onStartCommand(intent, flags, startId);
    }

    /**
     * 更新天气信息
     */
    private void updateWeather(){
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
        String weatherString = prefs.getString("weather", null);
        if (weatherString != null) {
            // 有缓存时直接解析天气数据
            Weather weather = Utility.handleWeatherResponse(weatherString);
            String weatherId = weather.basic.weatherId;

            String weatherUrl = "http://guolin.tech/api/weather?cityid=" +
```

```
                weatherId + "&key=bc0418b57b2d4918819d3974ac1285d9";
            HttpUtil.sendOkHttpRequest(weatherUrl, new Callback() {
                @Override
                public void onResponse(Call call, Response response) throws
                    IOException {
                    String responseText = response.body().string();
                    Weather weather = Utility.handleWeatherResponse(responseText);
                    if (weather != null && "ok".equals(weather.status)) {
                        SharedPreferences.Editor editor = PreferenceManager.
                            getDefaultSharedPreferences(AutoUpdateService.this).
                                edit();
                        editor.putString("weather", responseText);
                        editor.apply();
                    }
                }

                @Override
                public void onFailure(Call call, IOException e) {
                    e.printStackTrace();
                }
            });
        }
    }

    /**
     * 更新必应每日一图
     */
    private void updateBingPic() {
        String requestBingPic = "http://guolin.tech/api/bing_pic";
        HttpUtil.sendOkHttpRequest(requestBingPic, new Callback() {
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                String bingPic = response.body().string();
                SharedPreferences.Editor editor = PreferenceManager.getDefault
                    SharedPreferences(AutoUpdateService.this).edit();
                editor.putString("bing_pic", bingPic);
                editor.apply();
            }

            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }
        });
    }

}
```

可以看到，在 `onStartCommand()`方法中先是调用了 `updateWeather()`方法来更新天气，然后调用了 `updateBingPic()`方法来更新背景图片。这里我们将更新后的数据直接存储到 SharedPreferences 文件中就可以了，因为打开 WeatherActivity 的时候都会优先从 SharedPreferences 缓存中读取数据。

之后就是我们学习过的创建定时任务的技巧了，为了保证软件不会消耗过多的流量，这里将时间间隔设置为 8 小时，8 小时后 AutoUpdateReceiver 的 onStartCommand()方法就会重新执行，这样也就实现后台定时更新的功能了。

不过，我们还需要在代码某处去激活 AutoUpdateService 这个服务才行。修改 WeatherActivity 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {

    ...

    /**
     * 处理并展示Weather实体类中的数据。
     */
    private void showWeatherInfo(Weather weather) {
        ...
        weatherLayout.setVisibility(View.VISIBLE);
        Intent intent = new Intent(this, AutoUpdateService.class);
        startService(intent);
    }
}
```

可以看到，这里在 showWeatherInfo()方法的最后加入启动 AutoUpdateService 这个服务的代码，这样只要一旦选中了某个城市并成功更新天气之后，AutoUpdateService 就会一直在后台运行，并保证每 8 小时更新一次天气。

现在可以再提交一下代码：

```
git add .
git commit -m "增加后台自动更新天气的功能。"
git push origin master
```

14.8 修改图标和名称

目前的酷欧天气看起来还不太像是一个正式的软件，为什么呢？因为都还没有一个像样的图标呢。一直使用 Android Studio 自动生成的图标确实不太合适，是时候需要换一下了。

这里我事先准备好了一张图片来作为软件图标，由于我也不是搞美术的，因此图标设计得非常简单，如图 14.30 所示。



图 14.30 酷欧天气的图标

理论上来讲，我们应该给这个图标提供几种不同分辨率的版本，然后分别放入到相应分辨率的 mipmap 目录下，这里简单起见，我就都使用同一张图了。将这张图片命名成 logo.png，放入到所有以 mipmap 开头的目录下，然后修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.coolweather.android">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:name="org.litepal.LitePalApplication"
        android:allowBackup="true"
        android:icon="@mipmap/logo"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
    </application>

</manifest>
```

这里将`<application>`标签的 `android:icon` 属性指定成@mipmap/logo 就可以修改程序图标了。接下来我们还需要修改一下程序的名称，打开 res/values/string.xml 文件，其中 app_name 对应的就是程序名称，将它修改成酷欧天气即可，如下所示：

```
<resources>
    <string name="app_name">酷欧天气</string>
</resources>
```

现在重新运行一遍程序，这时观察酷欧天气的桌面图标，如图 14.31 所示。



图 14.31 手机桌面图标

养成良好的习惯，仍然不要忘记提交代码。

```
git add .
git commit -m "修改程序图标和名称。"
git push origin master
```

这样我们就终于大功告成了！

14.9 你还可以做的事情

经过五个阶段的开发，酷欧天气已经是一个完善、成熟的软件了吗？嘿嘿，还差得远呢！现在的酷欧天气只能说是具备了一些最基本的功能，和那些商用的天气软件比起来还有很大的差

距，因此你仍然还有非常巨大的发挥空间来对它进行完善。

比如说以下功能是你可以考虑加入到酷欧天气中的。

- 增加设置选项，让用户选择是否允许后台自动更新天气，以及设定更新的频率。
- 优化软件界面，提供多套与天气对应的图片，让程序可以根据不同的天气自动切换背景图。
- 允许选择多个城市，可以同时观察多个城市的天气信息，不用来回切换。
- 提供更加完整的天气信息，目前我们只使用了和风天气返回的一小部分数据而已。

另外，由于酷欧天气的源码已经托管在了 GitHub 上面，如果你想在现有代码的基础上继续对这个项目进行完善，就可以使用 GitHub 的 Fork 功能。

首先登录你自己的 GitHub 账号，然后打开酷欧天气版本库的主页：https://github.com/guolindev/coolweather，这时在页面头部的最右侧会有一个 Fork 按钮，如图 14.32 所示。

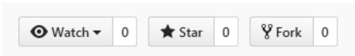


图 14.32　GitHub Fork 按钮

点击一下 Fork 按钮就可以将酷欧天气这个项目复制一份到你的账号下，再使用 `git clone` 命令将它克隆到本地，然后你就可以在现有代码的基础上随心所欲地添加任何功能并提交了。

第 15 章

最后一步——将应用发布到 360 应用商店

应用已经开发出来了，下一步我们需要思考推广方面的工作。那么如何才能让更多的用户知道并使用我们的应用程序呢？在手机领域，最常见的做法就是将程序发布到某个应用商店中，这样用户就可以通过商店找到我们的应用程序，然后轻松地进行下载和安装。

说到应用商店,在 Android 领域真的可以称得上是百家争鸣,除了谷歌官方推出的 Google Play 之外，在中国还有像 360、豌豆荚、百度、应用宝等知名的应用商店。当然，这些商店所提供的功能都是比较类似的，发布应用的方法也大同小异，因此这里我们就只学习如何将应用发布到 360 应用商店，其他应用商店的发布方法相信你完全可以自己摸索出来。

15.1 生成正式签名的 APK 文件

之前我们一直都是通过 Android Studio 来将程序安装到手机上的，而它背后实际的工作流程是，Android Studio 会将程序代码打包成一个 APK 文件，然后将这个文件传输到手机上，最后再执行安装操作。Android 系统会将所有的 APK 文件识别为应用程序的安装包，类似于 Windows 系统上的 EXE 文件。

但并不是所有的 APK 文件都能成功安装到手机上，Android 系统要求只有签名后的 APK 文件才可以安装，因此我们还需要对生成的 APK 文件进行签名才行。那么你可能会有疑问了，直接通过 Android Studio 来运行程序的时候好像并没有进行过签名操作啊，为什么还能将程序安装到手机上呢？这是因为 Android Studio 使用了一个默认的 keystore 文件帮我们自动进行了签名。点击 Android Studio 右侧工具栏的 Gradle→项目名→:app→Tasks→android，双击 signingReport，结果如图 15.1 所示。

```
Variant: debug
Config: debug
Store: C:\Users\Administrator\.android\debug.keystore
```

图 15.1　查看默认的 keystore 文件

也就是说，我们所有通过 Android Studio 来运行的程序都是使用了这个 debug.keystore 文件来进行签名的。不过这仅仅适用于开发阶段而已，现在酷欧天气已经快要发布了，要使用一个正式的 keystore 文件来进行签名才行。下面我们就来学习一下，如何生成一个带有正式签名的 APK 文件。

15.1.1　使用 Android Studio 生成

先学习一下如何使用 Android Studio 来生成正式签名的 APK 文件。点击 Android Studio 导航栏上的 Build→Generate Signed APK，首次点击可能会提示让我们输入操作系统的密码，如图 15.2 所示。



图 15.2　输入操作系统密码提示框

输入密码之后点击 OK，则会弹出如图 15.3 所示的创建签名 APK 对话框。



图 15.3　创建签名 APK 对话框

由于目前我们还没有一个正式的 keystore 文件，所以应该点击 Create new 按钮，然后会弹出一个新的对话框来让我们填写创建 keystore 文件所必要的信息。根据自己的实际情况进行填写就行了，如图 15.4 所示。

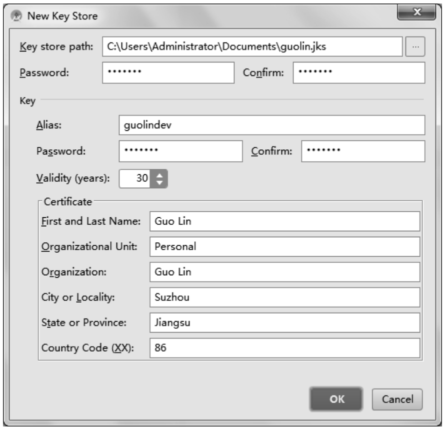


图 15.4 填写 keystore 文件信息

这里需要注意，在 Validity 那一栏填写的是 keystore 文件的有效时长，单位是年，一般建议时间可以填得长一些，比如我填了 30 年。然后点击 OK，这时我们刚才填写的信息会自动填充到创建签名 APK 对话框当中，如图 15.5 所示。

如果你希望以后都不用再输 keystore 的密码了，可以将 Remember passwords 选项勾上。然后点击 Next，这时就要选择 APK 文件的输出地址了，如图 15.6 所示。

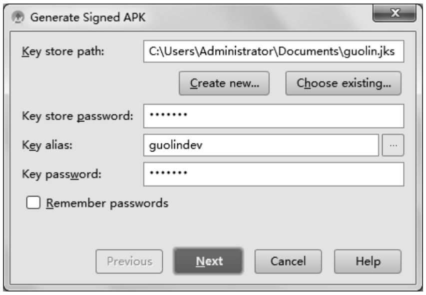


图 15.5 信息自动填充完整

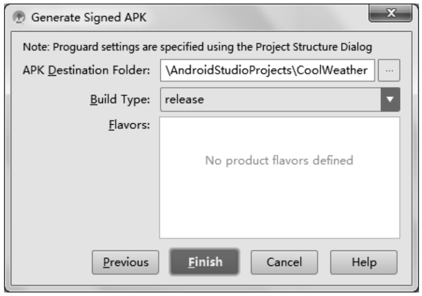


图 15.6 选择 APK 文件的输出地址

这里默认是将 APK 文件生成到项目的根目录下，我就不做修改了。现在点击 Finish，然后稍等一段时间， APK 文件就都会生成好了，并且会在右上角弹出一个如图 15.7 所示的提示。

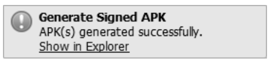


图 15.7 提示 APK 文件生成成功

我们点击提示上的 Show in Explorer 可以立刻查看生成的 APK 文件，如图 15.8 所示。

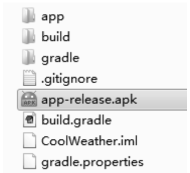


图 15.8　查看生成的 APK 文件

这里的 app-release.apk 就是带有正式签名的 APK 文件了。

15.1.2　使用 Gradle 生成

上一小节中我们使用了 Android Studio 提供的可视化工具来生成带有正式签名的 APK 文件，除此之外，Android Studio 其实还提供了另外一种方式——使用 Gradle 生成，下面我们就来学习一下。

Gradle 是一个非常先进的项目构建工具， 在 Android Studio 中开发的所有项目都是使用它来构建的。在之前的项目中，我们也体验过了 Gradle 带来的很多便利之处，比如说当需要添加依赖库的时候不需要自己再去手动下载了，而是直接在 dependencies 闭包中添加一句引用声明就可以了。

不过这里我要提醒你一句，如果你想将 Gradle 完全精通的话，这个难度就比较大了。Gradle 的用法极为丰富，想要完全掌握它的用法，其复杂程度并不亚于学习一门新的语言（Gradle 是使用 Groovy 语言编写的）。而 Android 中主要只是使用 Gradle 来构建项目而已，因此这里我们掌握一些它的基本用法就好了，重点还是要放在功能开发上面，不要本末倒置了。当然，如果你对 Gradle 非常感兴趣，也可以到网上去查询它的更多用法。

下面我们开始学习如何使用 Gradle 来生成带有正式签名的 APK 文件。编辑 app/build.gradle 文件，在 android 闭包中添加如下内容：

```
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.coolweather.android"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    signingConfigs {
        config {
            storeFile file('C:/Users/Administrator/Documents/guolin.jks')
            storePassword '1234567'
            keyAlias 'guolindev'
            keyPassword '1234567'
        }
```

```
        }
        buildTypes {
            release {
                minifyEnabled false
                proguardFiles getDefaultProguardFile('proguard-android.txt'),
                    'proguard-rules.pro'
            }
        }
    }
```

可以看到，这里在 android 闭包中添加了一个 signingConfigs 闭包，然后在 signingConfigs 闭包中又添加了一个 config 闭包。接着在 config 闭包中配置 keystore 文件的各种信息，storeFile 用于指定 keystore 文件的位置，storePassword 用于指定密码，keyAlias 用于指定别名，keyPassword 用于指定别名密码。

将签名信息都配置好了之后，接下来只需要在生成正式版 APK 的时候去应用这个配置就可以了。继续编辑 app/build.gradle 文件，如下所示：

```
android {
    ...
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
            signingConfig signingConfigs.config
        }
    }
}
```

这里我们在 buildTypes 下面的 release 闭包中应用了刚才添加的签名配置，这样当生成正式版 APK 文件的时候就会自动使用我们刚才配置的签名信息来进行签名了。

现在 build.gradle 文件已经配置完成，那么我们如何才能生成 APK 文件呢？其实非常简单，Android Studio 中内置了很多的 Gradle Tasks，其中就包括了生成 APK 文件的 Task。点击右侧工具栏的 Gradle→项目名→:app→Tasks→build，如图 15.9 所示。

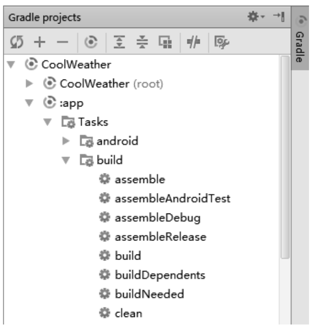


图 15.9 查看内置 Gradle Tasks

其中 assembleDebug 用于生成测试版的 APK 文件，assembleRelease 用于生成正式版的 APK 文件，assemble 用于同时生成测试版和正式版的 APK 文件。在生成 APK 之前，先要双击 clean 这个 Task 来清理一下当前项目，然后双击 assembleRelease，结果如图 15.10 所示。

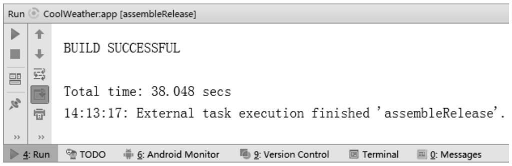


图 15.10　assembleRelease 执行成功

可以看到，这里提示我们 BUILD SUCCESSFUL，说明 assembleRelease 执行成功了。APK 文件会自动生成在 app/build/outputs/apk 目录下，如图 15.11 所示。

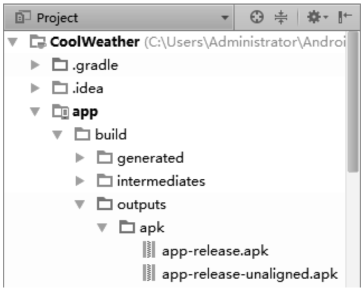


图 15.11　查看生成的 APK 文件

其中，app-release.apk 就是带有正式签名的 APK 文件了。另外还有一个 app-release-unaligned.apk，这是一个没有经过对齐的正式版 APK 文件，我们直接忽略它就可以了。

虽说现在 APK 文件已经成功生成了，不过还有一个小细节需要注意一下。目前 keystore 文件的所有信息都是以明文的形式直接配置在 build.gradle 中的，这样就不太安全。Android 推荐的做法是将这类敏感数据配置在一个独立的文件里面，然后再在 build.gradle 中去读取这些数据。

下面我们来按照这种方式实现。Android Studio 项目的根目录下有一个 gradle.properties 文件，它是专门用来配置全局键值对数据的，我们在 gradle.properties 文件中添加如下内容：

```
KEY_PATH=C:/Users/Administrator/Documents/guolin.jks
KEY_PASS=1234567
ALIAS_NAME=guolindev
ALIAS_PASS=1234567
```

可以看到，这里将 keystore 文件的各种信息以键值对的形式进行了配置，然后我们在 build.gradle 中去读取这些数据就可以了。编辑 app/build.gradle 文件，如下所示：

```
android {
    ...
```

```
        signingConfigs {
            config {
                storeFile file(KEY_PATH)
                storePassword KEY_PASS
                keyAlias ALIAS_NAME
                keyPassword ALIAS_PASS
            }
        }
        ...
    }
```

这里只需要将原来的明文配置改成相应的键值，一切就完工了。这样直接查看 build.gradle 文件是无法看到 keystore 文件的各种信息的，只有查看 gradle.properties 文件才能看得到。然后我们只需要将 gradle.properties 文件保护好就行了，比如说将它从 Git 版本控制中排除。这样 gradle.properties 文件就只会保留在本地，从而也就不用担心 keystore 文件的信息会泄漏了。

15.1.3 生成多渠道 APK 文件

现在你已经掌握了两种生成带有正式签名的 APK 文件的方式，从简易程度上来讲，两种方式差不多，基本都还是比较简单的，选择使用哪一种全凭你自己的喜好。

现在 APK 文件已经生成好了，可能在大多数情况下，我们都只需要一个 APK 文件就足够了，不过本小节中我们再来讨论一种比较特殊的情况——生成多渠道 APK 文件。

在本章的开头就已经提到过，目前 Android 领域的应用商店非常多，不像苹果只有一个 App Store。当然我们完全可以使用同一个 APK 文件来上架不同的应用商店，但是如果你有一些特殊需求的话，比如说针对不同的应用商店渠道来定制不同的界面，这就比较头疼了。

传统情况下，开发这种差异性需求非常痛苦，通常需要维护多份代码版本，然后逐个打成相应渠道的 APK 文件。一旦有任何功能变更就苦不堪言，因为每份代码版本里面都需要逐个修改一遍。

幸运的是，现在 Android Studio 提供了一种非常方便的方法来应对这种差异性需求，极大程度地解决了之前版本维护困难的问题，下面我们就来学习一下。

比如说这里我们准备生成 360 和百度两个渠道的 APK 文件，那么修改 app/build.gradle 文件，如下所示：

```
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.coolweather.android"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
```

```
        }
        productFlavors {
            qihoo {
                applicationId "com.coolweather.android.qihoo"
            }
            baidu {
                applicationId "com.coolweather.android.baidu"
            }
        }
        ...
    }
```

可以看到，这里添加了一个 productFlavors 闭包，然后在该闭包中添加所有的渠道配置就可以了。注意 Gradle 中的配置规定不能以数字开头，因此这里我将 360 的渠道名配置成了 qihoo。渠道名的闭包中可以覆写 defaultConfig 中的任何一个属性，比如说这里将 applicationId 属性进行了覆写，那么最终生成的各渠道 APK 文件的包名也将各不相同。

接下来我们开始针对不同渠道编写差异性需求。在 app/src 目录下（main 的平级目录）新建一个 baidu 目录，然后在 baidu 目录下再新建 java 和 res 这两个目录，如图 15.12 所示。

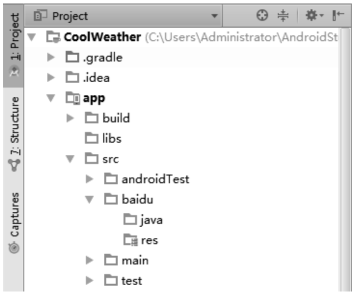


图 15.12　创建渠道专属目录

这样我们就可以在这里编写百度渠道特有的功能了，java 目录用于存放代码，res 目录用于存放资源，如果需要覆写 AndroidManifest 文件中的内容，还可以在 baidu 目录下再新建一个 AndroidManifest.xml 文件。

当然，实际上我们并没有什么渠道差异性的需求，因此这里也只是为了演示一下，我们就给不同渠道的 APK 起一个不同的应用名吧。

应用名之前是定义在 main/res/values/string.xml 文件中的，那么我们在 baidu 目录下也建立一个相同的目录结构，然后将 baidu/res/values/string.xml 中的内容进行如下修改：

```
<resources>
    <string name="app_name">酷欧百度版</string>
</resources>
```

这样百度渠道的 APK 就会使用 baidu/res/values/string.xml 中定义的应用名来覆盖原有的应用名。同样的道理，我们再新建一个 qihoo 目录，然后在 qihoo 目录下也建立相同的目录结构，并

将 string.xml 中的内容进行如下修改：

```
<resources>
    <string name="app_name">酷欧 360 版</string>
</resources>
```

这样我们就以一个简单的示例实现渠道差异性需求了，下面开始来生成多渠道的 APK 文件。观察右侧工具栏的 Gradle Tasks 列表，你会发现里面多出了几个新的 Task，如图 15.13 所示。

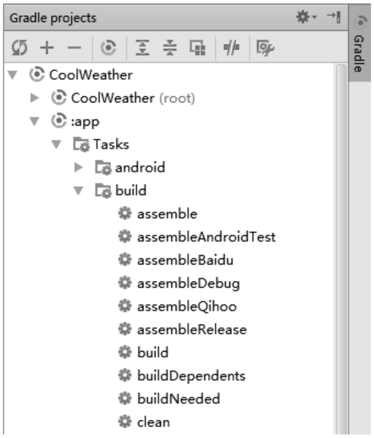


图 15.13　查看新的 Task

其中，如果你只想生成百度渠道的 APK 文件，那么就执行 assembleBaidu；如果你只想生成 360 渠道的 APK 文件，那么就执行 assembleQihoo；如果你想一次性生成所有渠道的 APK 文件，那么就还是执行 assembleRelease。

除了使用 Gradle 的方式生成之外，使用 Android Studio 提供的可视化工具也是能生成多渠道 APK 文件的，如图 15.14 所示。

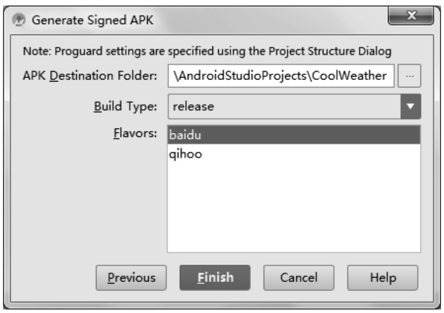


图 15.14　使用可视化工具生成多渠道 APK

这里我们可以选择是生成百度渠道的 APK 文件，还是生成 360 渠道的 APK 文件，如果你想

一次性生成多个渠道的 APK 文件，按住 CTRL 键就可以进行多选了。

接下来我们可以通过 `adb install` 命令将生成好的 APK 文件安装到模拟器上，如图 15.15 所示。



图 15.15　将生成的 APK 安装到模拟器上

`adb install` 命令的后面加上 APK 文件的路径，就可以将该 APK 文件安装到模拟器上了。我们使用同样的方法将百度和 360 这两个渠道的 APK 文件都安装到模拟器上，结果如图 15.16 所示。



图 15.16　模拟器上的安装结果

可以看到，目前模拟器上有 3 个版本的酷欧天气，这是由于之前我们在 productFlavors 中覆写了各渠道的 `applicationId` 属性，保证每个 APK 文件的包名都不相同，因而它们才能安装到同一个设备上面。另外，从应用名上来看，渠道差异性开发工作也顺利完成了。

不过，上面的例子只是为了演示生成多渠道 APK 功能而特意编写的，实际上我们并没有这个需求。现在将 `productFlavors` 闭包删除，恢复成之前的 APK 文件，我们准备进行上架操作。

15.2　申请 360 开发者账号

目前，酷欧天气的 APK 安装包已经准备好了，但如果想要把它发布到 360 应用商店，还需要去申请一个 360 开发者账号才行，申请地址是：http://dev.360.cn。

打开该网页，在页面顶部有登录和注册按钮。如果你还没有 360 账号，则需要在这里注册一个新的账号，如果你之前已经有 360 账号了，那么直接登录就可以了。

登录成功之后打开 http://dev.360.cn/mod/developer 这个网址，来申请成为开发者，如图 15.17 所示。



图 15.17　选择注册开发者类型

这里可以选择是申请成为个人开发者还是企业开发者。很显然，我们是以个人的身份来发布应用的，那么点击个人开发者就可以了。

接下来需要填写一些基本信息和联系方式，如图 15.18 所示。



图 15.18　填基本信息和联系方式

填写完基本信息之后向下滚动继续填写联系方式，全部填写完成之后，点击屏幕最下方的“同意并注册开发者”按钮来完成注册，如图 15.19 所示。



图 15.19　完成开发者注册

这样你就成功成为一名 360 开发者了！

15.3　发布应用程序

接下来我们开始发布酷欧天气这个应用，还是在浏览器访问地址：http://dev.360.cn，你会在界面上看到如图 15.20 所示的内容。

然后点击软件发布，就会显示如图 15.21 所示的界面。



图 15.20　软件发布和游戏发布



图 15.21　选择软件类型

我们需要选择是发布软件类应用还是电子书类应用，这里点击软件。接下来会弹出一个新的界面让我们上传 APK 以及填写应用信息。首先来上传 APK 吧，点击上传按钮，选择带有正式签名的 APK 文件，然后就会自动开始上传了，上传完成之后会显示如图 15.22 所示的界面。



图 15.22　上传 APK 完成

这个界面提醒我们，目前应用的安全系数较低，建议对 APK 进行加固。实际上这个是 360 应用商店的特殊需求，并不是所有应用商店都要求进行加固的。但是我们还是得按照它的要求来修改，不然审核可能会不通过。

这里点击立即加固按钮，360 会帮忙我们将原 APK 文件进行加固，并生成一个新的 APK 文件，如图 15.23 所示。

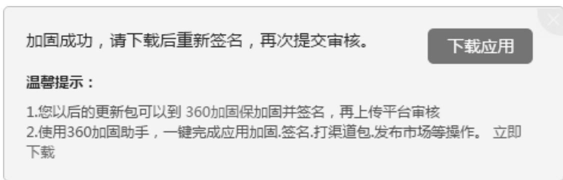


图 15.23　加固成功提示

不过这个加固后的 APK 文件是没有经过签名的，也就是说我们还需要将它下载下来，然后手动进行签名才行。

点击下载应用按钮，先将加固后的 APK 文件下载下来。接下来的工作就有点烦琐了，因为 Android Studio 中并没有提供对一个未签名的 APK 直接进行签名的功能，因此我们只能通过最原始的方式，使用 `jarsigner` 命令来进行签名。

在命令行界面按照以下格式输入签名命令：

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore [keystore 文件路径]
-storepass [keystore 文件密码] [待签名 APK 路径] [keystore 文件别名]
```

将`[]`中的描述替换成 keystore 文件的具体信息就能签名成功了，注意`[]`符号是不需要的。接着我们将签名后的 APK 文件重新上传就可以了。

APK 上传成功之后，接下来需要选择应用的分类，如图 15.24 所示。



图 15.24　选择应用分类

这里我们将应用分类选择成实用工具→天气。下面还有一个上传版权证明的选项，这是一个选填项，我们直接忽略就可以了。

接着向下滚动网页，设置支持的语言以及资费类型，如图 15.25 所示。



图 15.25　设置支持语言和资费类型

继续滚动网页，下面需要填写应用简介以及当前版本介绍，如图 15.26 所示。

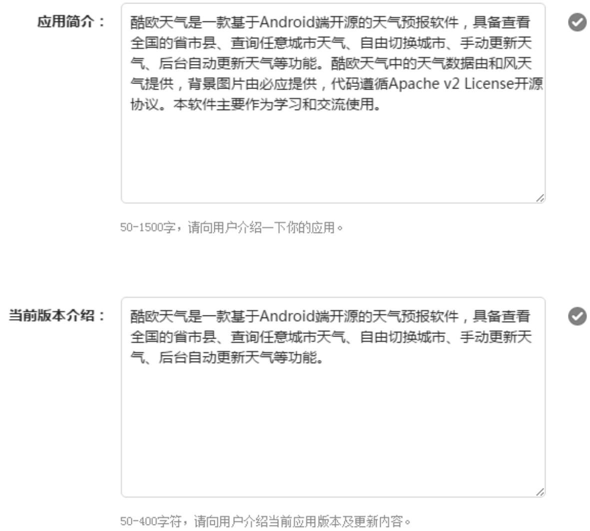


图 15.26　填写应用简介和当前版本介绍

在版本介绍的下面，360 还要求填写一项隐私权限说明，由于酷欧天气只申请了一个网络权限，因此没什么需要说明的，我们直接忽略这一项就可以了。

继续向下滚动网页，接下来需要上传一张高分辨率的应用图标，图标要求是 512×512 像素的 PNG 格式图片，如图 15.27 所示。



图 15.27　上传高分辨率的应用图标

上传好了图标，我们还需要提供 5 张酷欧天气的屏幕截图，点击上传截图按钮，然后选择准备好的图片即可，如图 15.28 所示。



图 15.28　上传屏幕截图

继续向下滚动，还有一个审核辅助说明的选填项，我们也直接忽略就可以了。最后就是一些额外的定制选项，如图 15.29 所示。



图 15.29　额外的辅助选项

这里我们选择不进行云测试，并在审核后立即发布。

激动人心的时刻终于到了，现在点击一下提交审核按钮就可以将酷欧天气发布到 360 应用商店了，这时会显示如图 15.30 所示的提示。

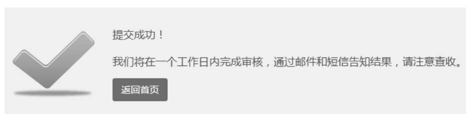


图 15.30　提交成功提示

由于 360 会对我们的应用程序进行审核，接下来又进入了等待当中。不过还好，根据提示来看，这次也许不需要等太久。

果不其然，过了几个小时之后在 360 手机助手上搜索酷欧天气关键字，就可以看到这个应用已经成功上线了，如图 15.31 所示。

点击进去可以查看应用的详情，如图 15.32 所示。



图 15.31　搜索酷欧天气关键字



图 15.32　查看应用详情

到了这里，我们就将应用程序的发布工作全部完成了，之后你应该尽可能地多为你的应用进行宣传，因为用户越多，你能得到的回报就越大。那么如何才能从我们辛辛苦苦编写的程序中得到回报呢？方式有很多种，其中较为常见的做法就是通过广告来进行盈利，因此下一节我们就学习一下，如何在应用程序中嵌入广告。

15.4　嵌入广告进行盈利

谷歌充分考虑到了可以在 Android 应用程序中嵌入广告来让开发者获得收入，因此早早地就收购了 AdMob 公司。AdMob 创立于 2006 年，是全球最早致力于在移动设备上提供广告服务的公司之一，如今成为了谷歌的子公司，AdMob 的广告更加适合在 Android 系统以及 Google Play 上面进行投放。

不过对于国内开发者来说，AdMob 可能就不是那么适合了。因为 AdMob 平台上的广告大多都是英文的，中文广告数有限，并且将 AdMob 账户中的钱提取到银行账户中也比较麻烦，因此这里我们就不准备使用 AdMob 了，而是将眼光放在一些国内的移动广告平台上面。在国内的这一领域，做得比较好的移动广告平台也不少，其中我个人认为腾讯广告联盟（原广点通）特别专业，因此我们就选择它来为酷欧天气提供广告服务吧。

15.4.1　注册腾讯广告联盟账号

下面开始动手，首先第一步我们需要注册一个腾讯广告联盟的账号，注册地址为：http://e.qq.

com/dev/index.html。

打开该网页，选择使用QQ号登录，然后就会自动跳转到腾讯广告联盟的注册界面，如图15.33所示。图15.33中的所有内容都是必填项，我们按照实际情况来填写就可以了，填写完成之后点击下一步，如图15.34所示。



图15.33　填写个人信息



图15.34　填写银行卡信息

由于腾讯广告联盟涉及提现服务，因此我们还需要填写银行卡信息，并上传银行卡照片。填写完成之后继续点击下一步，如图15.35所示。

最后，将你的身份证正反面照片上传，点击提交按钮，就能提交审核了，如图15.36所示。

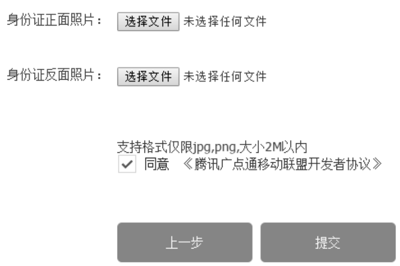


图15.35　上传身份证照片



图15.36　提交审核

只要你前面填写的内容都是真实有效的，审核一般都会很快通过，这里我们只需要耐心等待几个小时就好了。

15.4.2　新建媒体和广告位

审核通过之后，我们就可以进入到腾讯广告联盟的后台，开始给酷欧天气添加广告了。首先需要进入媒体管理界面，点击新建媒体按钮，这时会显示一个页面来让你填写应用的相关信息，我们根据提示一一填好即可，如图 15.37 所示。



图 15.37　填写应用的相关信息

注意这里需要填写一个详情页地址，也就是酷欧天气在 360 应用商店上的详情页地址。打开 http://zhushou.360.cn，在搜索框上输入“酷欧天气”，就能找到该地址了。

填写完成之后点击下一步，接下来需要下载 SDK，如图 15.38 所示。

点击 Android SDK 下载按钮，先把 SDK 下载下来，我们稍后就会进行接入。继续点击下一步，如图 15.39 所示。



图 15.38　下载 SDK



图 15.39　完成媒体创建

这里要求填入一个 APK 的下载的地址，我们直接就填入图 15.37 当中的详情页地址就可以

了。点击完成按钮，现在又会进入到审核等待当中。

为什么新建媒体也需要进行审核呢？这是因为腾讯为了防止某些开发者在垃圾软件上面投放广告，因此要求开发者必须提交应用程序的 APK 文件进行审核，只有审核通过的应用才允许进行广告投放。那么我们只能继续等待。审核通过之后，在媒体管理界面查看新建媒体的状态，如图 15.40 所示。



应用ID	媒体名称	联盟开通状态	业务状态	系统平台	操作
1105585573	酷欧天气	已开通	正常	Android	修改 新建广告位

图 15.40　查看新建媒体的状态

可以看到，联盟开通状态显示已开通，业务状态显示正常，说明新建的媒体已经通过审核了。注意这里还自动生成了一个应用 ID，我们稍后就会用到。

现在点击新建广告位，就可以来创建一个广告位了，如图 15.41 所示。



图 15.41　新建广告位

首先要输入广告位的名称，然后选择广告位的类型。腾讯广告联盟支持 Banner、应用墙、插屏和开屏这 4 种广告类型，具体每种广告类型的区别你可以通过查阅文档进行了解，这里我们选择开屏广告。接下来还可以对一些敏感行业的广告进行屏蔽，选择完成之后点击创建按钮完成广告位创建。

现在进入到广告位管理界面，就能查看到我们刚刚新建的广告位了，如图 15.42 所示。



名称&ID	广告位类型	所属应用&ID	业务状态	广告位状态	操作
启动广告 4010212448179536	开屏	酷欧天气 1105585573	正常	启用中	修改

图 15.42　查看新建广告位

其中，4010212448179536 是广告位 ID，1105585573 是应用 ID。有了这两个数据之后，我们就可以开始接入广告 SDK 了。

15.4.3　接入广告 SDK

首先将刚才下载的广告 SDK 压缩包解压，里面的内容非常简单，如图 15.43 所示。

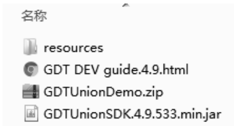


图 15.43　广告 SDK 压缩包中的内容

其中 resources 文件夹中放的是一些资源图片，我们使用不到。GDT DEV guide.4.9.html 是广告 SDK 的对接文档，GDTUnionDemo.zip 是广告 SDK 的对接示例，GDTUnionSDK.4.9.533.min.jar 则是广告 SDK 中最主要的一个 Jar 包文件了。

由于腾讯广告 SDK 中的功能还是挺多的，这里不可能面面俱到，将每一个功能都进行详细地讲解。因此我准备只讲解开屏广告这一种类型的广告用法，剩下的其他功能你可以通过阅读文档来进行学习。

我们先将 GDTUnionSDK.4.9.533.min.jar 复制到 app/libs 目录当中，并点击一下 Android Studio 顶部工具栏中的 Sync 按钮完成同步。

接着在 AndroidManifest.xml 中声明以下权限，其中网络访问权限是之前声明过的，不需要声明两遍。

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_UPDATES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

注意，其中 READ_PHONE_STATE、ACCESS_COARSE_LOCATION 和 WRITE_EXTERNAL_STORAGE 这 3 个权限是危险权限，因此我们待会还需要进行运行时权限处理。

接下来在<application>标签中添加如下内容：

```
<activity
    android:name="com.qq.e.ads.ADActivity"
    android:configChanges="keyboard|keyboardHidden|orientation|screenSize" />
<service
    android:name="com.qq.e.comm.DownloadService"
    android:exported="false" />
```

这样就将配置工作完成了。

然后我们还需要创建一个用于显示开屏广告的活动，右击 com.coolweather.android 包→New→Activity→Empty Activity，创建一个 SplashActivity，并将布局名指定成 activity_splash.xml。修改 activity_splash.xml 中的代码，如下所示：

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</RelativeLayout>
```

这里只有一个空的 RelativeLayout，我们并不需要在 RelativeLayout 当中放入什么内容，但是必须给它定义一个 id。

接着修改 SplashActivity 中的代码，如下所示：

```
public class SplashActivity extends AppCompatActivity {

    private RelativeLayout container;

    /**
     * 用于判断是否可以跳过广告，进入 MainActivity
     */
    private boolean canJump;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        container = (RelativeLayout) findViewById(R.id.container);
        // 进行运行时权限处理
        List<String> permissionList = new ArrayList<>();
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_PHONE_
            STATE) != PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.READ_PHONE_STATE);
        }
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_
            COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.ACCESS_COARSE_LOCATION);
        }
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE_
            EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.WRITE_EXTERNAL_STORAGE);
        }
        if (!permissionList.isEmpty()) {
            String [] permissions = permissionList.toArray(new String[permissionList.
                size()]);
            ActivityCompat.requestPermissions(this, permissions, 1);
        } else {
            requestAds();
        }
    }

    /**
     * 请求开屏广告
     */
    private void requestAds() {
        String appId = "1105585573";
```

```
        String adId = "4010212448179536";
        new SplashAD(this, container, appId, adId, new SplashADListener() {
            @Override
            public void onADDismissed() {
                // 广告显示完毕
                forward();
            }

            @Override
            public void onNoAD(int i) {
                // 广告加载失败
                forward();
            }

            @Override
            public void onADPresent() {
                // 广告加载成功
            }

            @Override
            public void onADClicked() {
                // 广告被点击
            }
        });
    }

    @Override
    protected void onPause() {
        super.onPause();
        canJump = false;
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (canJump) {
            forward();
        }
        canJump = true;
    }

    private void forward() {
        if (canJump) {
            // 跳转到 MainActivity
            Intent intent = new Intent(this, MainActivity.class);
            startActivity(intent);
            finish();
        } else {
            canJump = true;
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
       int[] grantResults) {
```

```
            switch (requestCode) {
                case 1:
                    if (grantResults.length > 0) {
                        for (int result : grantResults) {
                            if (result != PackageManager.PERMISSION_GRANTED) {
                                Toast.makeText(this, "必须同意所有权限才能使用本程序",
                                    Toast.LENGTH_SHORT).show();
                                finish();
                                return;
                            }
                        }
                        requestAds();
                    } else {
                        Toast.makeText(this, "发生未知错误", Toast.LENGTH_SHORT).show();
                        finish();
                    }
                    break;
                default:
            }
        }

    }
```

可以看到，在 onCreate()方法中，我们先是获取到了 RelativeLayout 的实例，紧接着就开始进行运行时权限处理。由于这里也是需要在运行时一次性申请多个权限，因此采用了和 11.3.2 小节同样的写法，相信你一定不会陌生吧。

当用户同意了所有的权限申请之后，就会调用 requestAds()方法来请求广告数据。在 requestAds()方法中我们先是定义了 appId 和 adId 这两个变量，它们的值就是在腾讯广告联盟后台生成的应用 ID 和广告位 ID，然后创建 SplashAD 的实例来获取广告数据。SplashAD 的构造函数接收 5 个参数，第 1 个参数是当前活动的实例，第 2 个参数是 RelativeLayout 的实例，第 3 个参数是应用 ID，第 4 个参数是广告位 ID，相信前 4 个参数都没什么需要解释的。第 5 个参数是一个 SplashADListener 的实例，用于监听广告数据的回调。其中 onADDismissed()方法会在广告显示完毕时回调，onNoAD()方法会在广告加载失败时回调，onADPresent()方法会在广告加载成功时回调，onADClicked()方法会在广告被点击时回调。当广告显示完毕或者广告加载失败时，我们调用 forward()方法跳转到 MainActivity，并将当前活动关闭即可。

另外注意这里还使用了一个 canJump 变量用于对活动跳转进行控制。这是因为如果用户点击了广告，会启动一个新的活动来展示广告的详细内容，这个时候即使回调了 onADDismissed()方法，显然也不应该启动 MainActivity，因此我们在 onPause()方法中将 canJump 设置成了 false。然后在 forward()方法中发现 canJump 是 false，因此不会进行跳转，但是会将 canJump 设置成 true。最后，当用户看完了广告回到 SplashActivity 时，onResume()方法将会执行，这个时候发现 canJump 是 true，因此就会调用 forward()方法来启动 MainActivity。

整体流程大概就是这个样子了，接下来我们还需要将主活动设置成 SplashActivity 而不再是 MainActivity，否则广告界面将无法得到展示。修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.coolweather.android">
    ...
    <application
        android:name="org.litepal.LitePalApplication"
        android:allowBackup="true"
        android:icon="@mipmap/logo"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
        </activity>
        <activity android:name=".SplashActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        ...
    </application>

</manifest>
```

这样我们就将广告 SDK 全部对接完成了，现在可以重新运行一下程序来看一看效果。不过需要注意，广告在模拟器上是不会显示的，我们要用真正的手机测试才行。程序启动后首先会弹出运行时权限的申请对话框，全部都点击允许之后就能看到广告数据了，如图 15.44 所示。



图 15.44　显示广告数据

开屏广告会持续 5 秒钟时间，然后就会自动跳转到 MainActivity 中，后面的流程就和之前是完全一样的了。

15.4.4　重新发布应用程序

现在我们已经成功在酷欧天气中接入了广告功能，那么是时候将这个新版本更新到 360 应用商店上了。

由于即将发布的会是新一版的酷欧天气，因此在生成安装包之前还需要修改一下应用程序的版本号信息。编辑 app/build.gradle 文件，如下所示：

```
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.coolweather.android"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 2
        versionName "1.1"
    }
    ...
}
```

可以看到，这里将 versionCode 改成了 2，versionName 改成了 1.1。需要注意的是，每个版本的 versionCode 和 versionName 都不能和其他版本相同，且新版应用的版本号必须大于老版应用的版本号。

接下来我们就可以使用在 15.1 节学习的技术来生成新的 APK 文件，具体的步骤就不再重复介绍了，最终会生成一个新的 app-release.apk 文件，下面我们将它重新发布到 360 应用商店。

打开 360 开发者后台的管理中心页面，然后点击酷欧天气的更新管理按钮，如图 15.45 所示。

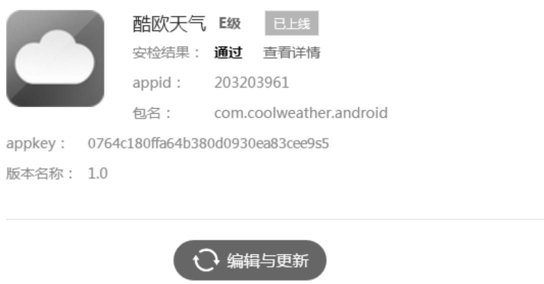


图 15.45　更新酷欧天气

点击编辑与更新按钮，就能上传新的 APK 文件了。注意上传之后仍然会提醒应用的安全系数较低，我们只需要使用和之前同样的方式进行加固就可以了。

另外，由于新版的酷欧天气中增加了一些敏感隐私权限，因此我们还需要在这一项上面做出说明，如图 15.46 所示。

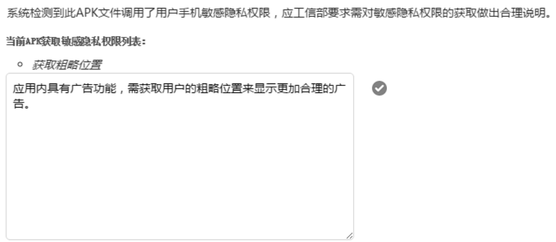


图 15.46　对敏感隐私权限进行说明

现在只需要点击页面最下方的提交审核按钮，新版本的酷欧天气就发布成功了，当然还需要通过 360 的审核才行。以后每当有用户观看或点击了应用程序中的广告时，我们就能真正地得到收益。在腾讯广告联盟的后台管理界面可以查看到每天的收益情况，如图 15.47 所示。



图 15.47　查看广告收益情况

你的应用越成功，所获得的广告收益也会越多，因此赶快去编写更多优秀的应用程序来赚更多的钱吧，相信通过整本书的学习，你已经有足够的能力做到了！

15.5　结束语

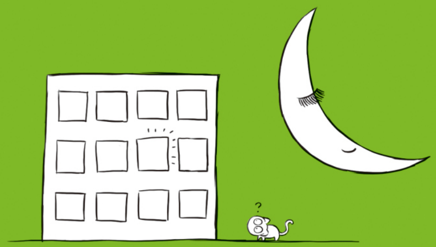
就这样，本书所有的内容你都学完了，现在你已经成功毕业，并且成为了一名合格的 Android 开发者。但是如果想要成为一名出色的 Android 开发者，光靠本书中的这些理论知识以及少量的实践还是不够的，你需要真正步入到工作岗位当中，通过更多的项目实战来不断地历练和提升自己。

唠叨了整本书的话，但是到了最后却不知道该说点什么好，我不想说我能教你的就只有这些了，因为实际上我想教你或者和你一起探讨的内容还有很多很多，不过限于篇幅的原因，本书的内容就只能到此为止了。但我会长期在博客和微信公众号上面分享更多 Android 相关的技术文章，如果感兴趣的话，可以到我的博客和公众号中继续学习。当然，如果对本书中的内容有疑问，也可以到博客或公众号中给我留言，我的博客地址如下：

http://guolin.tech

扫一扫下方二维码即可关注我的公众号：

好了，就到这里吧，祝愿你未来的 Android 之旅都能愉快。



我挥舞着键盘和本子，
发誓要把世界写个明明白白。

图灵社区：iTuring.cn
热线：(010) 51095186 转 600

分类建议 计算机／移动开发／程序设计

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-43978-9

ISBN 978-7-115-43978-9
定价：79.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱:
ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks